SENSOR NETWORK INTEROPERABILITY AND RECONFIGURATION THROUGH MOBILE AGENTS

A Dissertation in
Computer Science and Engineering

by
Evens Jean

© 2011 Evens Jean

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

May 2011
The thesis of Evens Jean was reviewed and approved* by the following:

Ali R. Hurson  
Professor of Computer Science and Engineering  
Dissertation Adviser  
Co-Chair of Committee

Robert T. Collins  
Associate Professor of Computer Science and Engineering  
Co-Chair of Committee

Vijaykrishnan Narayanan  
Professor of Computer Science and Engineering

Ling Rothrock  
Associate Professor of Industrial and Manufacturing Engineering

Yu (Cathy) Jiao  
Research Scientist at Oak Ridge National Laboratory  
Special Member

Dr. Raj Acharya  
Professor of Computer Science and Engineering  
Head of the Department of Computer Science and Engineering

*Signatures are on file in the Graduate School
Sensor networks generally consist of small devices deployed in an area to perform a task through coordination and communication. The current paradigm in sensor network promotes isolated networks with statically tasked nodes; this hinders the potential for applications to harness the power of fused data from multiple networks. Furthermore, while individual networks subsist in a dynamic environment, the tasks of the nodes are generally static and cannot adapt to changes in application requirements. To address these issues, this dissertation aims at rendering sensor networks interoperable and reconfigurable through reliance on mobile agent technology. The proposed approach intends to leverage data collected from individual networks to increase the set of knowledge available to applications. Our research has led to the introduction of a Reconfigurable and Interoperable Sensor Network (RISN) architecture that relies on mobile agents to achieve the stated goal. RISN provides network interoperability, and supports dynamic tasking of nodes through agent migration. The system further relies on Field Programmable Gate Arrays (FPGA) to handle potential issues associated with the execution time and latency of results by allowing applications to take advantage of hardware accelerators. The use of agent technology introduces security risks to the system; as such, we also investigated the security issues that plague agent-based systems. As the use of agents in practical applications is primarily limited by the security concerns of hosts, our investigation resulted in the introduction of a Distributive and Adaptive Security-Monitoring through Agent Collaboration (DASAC) to address such concerns. DASAC classifies agents based on their execution patterns on current and prior hosts in order to thwart attacks thereby abating the security concerns of hosts. Lastly, as a proof-of-concept, we discuss the implementation of a distributed target-tracking application that relies on the interoperability and reconfiguration of networks that RISN affords to applications, in order
to continuously maintain the location of object of interest despite the heterogeneity of the network, and limited views of any one sensor.
# TABLE OF CONTENTS

**CHAPTER 1** INTRODUCTION .................................................................................. 1

**CHAPTER 2** BACKGROUND AND RELATED WORK........................................... 6

2.1 MOBILE AGENT TECHNOLOGY ........................................................................ 6

2.1.1 Mobile Agent Security .................................................................................. 7

2.1.2 Mobile Agent Platforms ............................................................................... 11

2.2 SENSOR NETWORK ............................................................................................ 12

2.2.1 Sensor Network Applications ....................................................................... 13

2.2.1.1 Military Applications ................................................................................ 14

2.2.1.2 Environmental Monitoring ....................................................................... 14

2.2.1.3 Home and Office Applications .................................................................. 14

2.2.1.4 Habitat Monitoring .................................................................................. 14

2.2.1.5 Health Monitoring Applications ................................................................ 15

2.2.1.6 Others ...................................................................................................... 15

2.3 TARGET TRACKING IN SENSOR NETWORKS ................................................. 15

2.4 DYNAMIC TASKING IN SENSOR NETWORKS ............................................... 18

2.4.1 Active Sensors .............................................................................................. 19

2.4.1.1 Mate ......................................................................................................... 19

2.4.1.2 SensorWare ............................................................................................. 21

2.4.1.3 Deluge ..................................................................................................... 23

2.4.1.4 Agilla ........................................................................................................ 24

2.4.1.5 UC Davis Mobile Agent Framework ......................................................... 25

2.4.1.6 ActorNet .................................................................................................. 26

2.4.1.7 Reconfigurable Sensor Network with SOS .............................................. 27

2.4.2 FPGA-based Sensor Nodes .......................................................................... 28
LIST OF FIGURES

Figure 3-1: Lifecycle states of Aglets ................................................................. 39
Figure 3-2: Overview of DASAC Implementation in SAS .............................. 50
Figure 3-3: Training Accuracy of hosts ............................................................. 54
Figure 3-4: Local Vs. DASAC Accuracy ............................................................... 56
Figure 3-5: Effect of trust levels on DASAC Accuracy ................................. 57
Figure 3-6: False Alarm Rate, Miss Alarm Rate and Accuracy Rate ............ 60
Figure 3-7: Processing Vs. Communication Time of Arriving Agents .......... 60
Figure 4-1: Reconfigurable and Interoperable Sensor Network .................. 67
Figure 4-2: Components of RISN Overlay Node ........................................... 68
Figure 4-3: Implementation of Mean-Shift Algorithm .................................. 77
Figure 4-4: IIS Visualization .............................................................................. 84
Figure 4-5: FPGA Resource Utilization .............................................................. 88
Figure 4-6: Communication Cost ..................................................................... 91
Figure 4-7: IIS Computational Efficiency ......................................................... 93
Figure 4-8: LLT Computational Efficiency (Multiplication) ......................... 94
Figure 4-9: LLT Computational Efficiency (Division) ..................................... 95
Figure 4-10: Energy Usage for Multiplication .................................................. 96
Figure 4-11: Theoretical Effect of Percentage of RISN Hardware Modules’ Dynamic Power ................................................................. 96
Figure 5-1: Experimental Objects Layout ......................................................... 105
Figure 5-2: Distributed Tracking Algorithm ..................................................... 108
Figure 5-3. Overview of Tracking System ......................................................... 109
Figure 5-4. Results of Distributed Tracking System ....................................... 110
Figure 5-5. Bandwidth adjustments .................................................................. 111
LIST OF TABLES

Table 2-1: Counter-Measures to deter security threats to Agent systems..........................10
Table 2-1: Activation Methods and Associated Characteristics .........................................16
Table 2-3: Comparison of Approaches to Sensor Network Re-Configurability ....................30
Table 2-4: Evaluation of Interoperable WSN Proposals .....................................................34
Table 3-1: Classification Features for Each Agent ............................................................52
Table 4-1: Estimated Power Usage ..................................................................................92
ACKNOWLEDGEMENTS

I am very thankful for the leadership, support, guidance and hard work from my advisors, Dr. Ali R. Hurson and Dr. Robert T. Collins, along with the members of my committee: Dr. Yu (Cathy) Jiao, Dr. Vijay Narayanan, and Dr. Ling Rothrock. Many thanks to my fellow graduate students especially for all the discussions that have helped improve the quality of this work all the while making graduate school an enjoyable experience. The staff in the Computer Science and Engineering department has been really supportive over the course of my stay. I thank them all for putting up with my questions and me.

Over the years, my research has been supported by various grants such as the National Science Foundation under the contract IIS-0324835. The Laboratory Directed Research and Development Program of Oak Ridge National Laboratory (ORNL) managed by UT-Battelle, LLC, for the U.S. Department of Energy under Contract No. DE-AC05-00OR22725 has also supported this work, along with Xilinx through provision of tools and hardware without which this dissertation would not have been possible. I am also grateful for the Bunton-Waller Assistantship, the GE First-Year Faculty for the Future Fellowship in Engineering, the Graduate School Fellowship, and the Alfred P. Sloan Foundation Minority Ph.D. Fellowship, all of which have been instrumental in completing this work.

Last, but not least, I would like to thank my family and friends. Without their love and support, none of this would have been possible. At the end of everyday, they have always helped me recharge and get me ready for yet another day of fighting to accomplish this end result.
Chapter 1

Introduction

In a Sensor Network, several sensing nodes are utilized to form a network capable of reacting to environmental stimuli. The sensor nodes are generally low-power, low-memory devices with highly constrained computational capability. The data collected by the nodes are relayed to a special node in the network referred to as a base station or sink for processing. The increasing popularity of the use of sensor network in various applications is not without consequences, one of which is node proliferation. Traditional sensor network architectures advocate the deployment of statically tasked node forming an isolated network. The promotion of isolated networks with statically tasked nodes hinders the potential for applications to harness the power of fused data from multiple networks. The sensing ability of individual sensor networks in an environment may be limited to chemical, acoustic or video sensing. Coupled with the fact that the coverage area of individual networks may overlap, interoperation can drastically leverage the set of applications that can benefit from the data collected from individual networks. Consider a populated area, such as an airport, with multiple sensors deployed for pre-defined tasks including but not limited to analyzing the flow of human traffic, as well as detection of fire, and presence of banned materials. Chemical, acoustic and visual sensors may be at play in such an environment, but are forced to focus on static tasks whereas the data collected could be fused to form a more comprehensive view of the environment. Intelligent agents could be dispatched to an area where fire has been detected. The agents could rely on other sensors and retrieve heat signatures and views from surrounding cameras to determine if the fire was started intentionally as a diversion to facilitate criminal activities or if it is due to the smoke from a cigarette. The sprinkler system does not always have to be activated, nor do firefighters have to be called upon for every instance that a smoke alarm goes off. Response from emergency personnel could be more adequate to the
situation at hand. Development of such systems is highly dependent upon the ability of intelligent agents to access the data of interest regardless of proprietary issues, and heterogeneity of surrounding sensors. Allowing interoperability of networks could prevent node proliferation and provide applications with a richer set of data upon which to rely in accomplishing their tasks. Ultimately, sensor networks should be able to support a diverse array of tasks, depending on the needs of applications. To that end, the use of agents in sensor network has been introduced in the literature. However, such proposals have not envisioned the need for different networks to interoperate towards the accomplishment of a task.

The introduction of agents in the sensor network environment is not without repercussions. As an autonomous intelligent entity, agents can migrate to data sources of interest, retrieve, and intelligently process available information and make recommendations to their owners. One of the major hindrances to the widespread use of agents in applications stems from their inherent ability to migrate, hence, as a consequence, a wide range of security issues have surfaced. There are numerous security threats that face the agent paradigm. The majority of the proposals introduced thus far to secure agent applications has been theoretical, and has approached the issue from a centralized aspect attempting to protect either the agents or the hosts to which they migrate. In order to fully extract the benefits of agents in information retrieval, it is imperative that a secure environment be provided. Furthermore, it is of primordial importance that agent security be adequate to the environment in which the agents operate. By that we mean that agent security should not be a centralized issue since the agents exist in a distributed environment. Despite the security threats, the introduction of agents can bring flexibility to sensor network applications; the network can be made to adapt to changing observations and, or requirements while intelligently processing collected data.

This dissertation intends to introduce potential solutions to the aforementioned issues. To that end, a Reconfigurable and Interoperable Sensor Network (RISN) hardware-overlay
architecture to harness and process information from surrounding networks is introduced. RISN (read as risen) no longer views the network formed by various nodes deployed by an administrator as an isolated entity dedicated to one task. Instead, the system promotes the sharing of data from underlying networks to enable applications to take advantage of a more comprehensive view of the environment of interest. The system exploits the mobile agent paradigm to provide users with the ability to migrate tasks to locations of interest and reprogram nodes based on the current need. In order to deal with issues relating to latency of results, the system relies on Field Programmable Gate Arrays (FPGAs) to reduce the execution time of critical tasks. In essence, RISN provides:

- Increased processing power through the use of hardware accelerators to help reduce the latency of results and allow applications to meet their constraints.
- Interoperability with surrounding networks to prevent node proliferation while increasing the set of data that can be utilized by applications in gaining a comprehensive view of their environment.
- Dynamic tasking through the use of mobile agents to render nodes reprogrammable.
- Service provision to efficiently support common needs of applications.

Due to its reliance on FPGAs, we do recognize that the size of the RISN nodes may limit the application domain of the model. However, FPGAs are becoming increasingly smaller, and are presently available in sizes comparable to the Berkeley motes [19, 82, 83], which are quite popular in the research community. Furthermore, we expect RISN nodes to be pricier than traditional sensor nodes. The price difference will undoubtedly deter the use of the RISN overlay nodes in traditional sensor network environments, where it is generally assumed that nodes are very cheap and thousands are easily affordable by an individual or company for a specific use. However, as an overlaid network composed of computationally powerful nodes, we envision that only a few number of RISN nodes need to be deployed in an environment in order to achieve
RISN’s main goals. Being that the overlaid network will consist of a small set of nodes, collection of such nodes in the event of failure or to replace their batteries is feasible especially in urban settings. The use of hardware accelerators will enable applications to react faster to environmental stimuli which can prove to be crucial to time sensitive applications. Moreover, through the use of agents, the task of the network does not have to be pre-defined or static, the overlaid network as a whole can be reconfigured to adapt to changes in the underlying networks. The existence of interoperable and reprogrammable nodes coupled with the speedup in execution time of applications can help overcome the drawbacks of deploying RISN nodes.

By relying on the mobile agent paradigm to accomplish its goals, RISN opens itself to the security issues plaguing agent-based applications. Thusly, this thesis also undertakes the task of addressing the security issue of agents to foster their use in the underlying platform herein proposed and in information retrieval, in general. Mobile agents and the hosts on which they execute represent the two main entities in agent systems. While the threat of hosts attacking agents is serious, the converse has been identified as the main hindrance to the deployment of agent-based systems [1]. Relying on the assumption that a malicious agent is likely to attack more than one host in the system, our work introduces a boosting-based monitoring system that allows hosts to learn and classify agents through collaboration. The issue of protecting agents from hosts is also addressed through implementation of secured communication channels and the concept of read-only data in the agent platform used to evaluate our work.

RISN’s ultimate goal is to allow applications to rely on a comprehensive view of the environment of interest, while reprogramming nodes and taking advantage of hardware accelerators to accomplish the task at hand. In order to highlight the potential that RISN presents to applications, a vision-based tracking application has been implemented atop the RISN prototype. In general, tracking applications are interested in continuously maintaining the location of an object traversing a sensed environment. Such applications need to adapt to the object’s
motion as well as the heterogeneity of the sensing environment. By that we mean that the tracker should be able to adapt to the fact that an object may not be traceable at times from one or more nodes using a particular feature set. The location of the target could however still be determined through other feature sets or sensing abilities of the network such as heat or sound signatures. Drawing on that logic, the implemented distributed tracking applications relies on RISN to adapt to the environment, and maintain the location of the object of interest despite the heterogeneity of sub-networks through which the object may travel.

In short, this dissertation addresses issues of

- Mobile Agent security from both a distributed and centralized standpoint in order to hinder potentially malicious activities from disrupting the proper functioning of the system.
- Sensor Network interoperability and reconfiguration through RISN providing a service-based architecture and hardware accelerators in order to increase the amount of observations available to users while allowing for a reduction in the latency of results.
- Vision-based tracking in a heterogeneous environment through an ad-hoc agent network capable of maintaining the continuous locations of the object using a potentially diverse set of tracking features.

In further detailing our work, we first provide the reader with an overview of the current state of the art in chapter 2, detailing the concept of Mobile Agents, and discussing relevant works in sensor network and target tracking. Chapter 3 details our approach in securing the execution environment of agents. Chapter 4 introduces the proposed Reconfigurable and Interoperable Sensor Network (RISN) architecture representing the core of this thesis. Chapter 5 showcases our work in implementing a target tracking application serving as proof of concept of RISN potential in providing a dynamically tasked and interoperable sensor network environment. Finally, we drew conclusions in chapter 6 highlighting the future research in this area.
Chapter 2

Background and Related Work

The focus of this chapter is to introduce the reader to the research concepts that are fundamental to understanding the remainder of this dissertation. As such, we discuss mobile agent technology in Section 2.1; sensor network and target tracking in sensor network are presented in Sections 2.2, and 2.3, respectively. Section 2.4 introduces reconfigurable sensor networks, while Section 2.5 addresses sensor network interoperability. A summary of the chapter is provided in Section 2.6.

2.1 Mobile Agent Technology

The term “Mobile Agents” refers to a programming paradigm focused around the ability of a program to halt its execution, then move to a new environment where execution can then be resumed. In general, mobile agents are software entities that roam a network to carry out a task. These agents are typically mobile (though not a norm), autonomous, and perceivably intelligent. They can cooperate with one another to achieve a, not necessarily, common goal. The mobility of agents is not static; it can depend on current computation or be specified in advance by the user via an agent itinerary. The mobile agent system is composed of two primary components namely the execution environment provided by the hosts and the mobile agent that travels to various environments on a network [2].

Mobile Agents find their applications in environments where there is a need to collect data from multiple sources over a network. The use of mobile agents provides programmers with a new computational model that deviates from the traditional client-server approach. Through the
use of agents, the performance of a system can be significantly improved [3, 4], as agents take the computation to the data thereby reducing network traffic. The ability for a user to dispatch an agent to roam a network in search of travel tickets has been cited in the research community [5] as one possible application of this programming paradigm. After deployment, such an agent would then be able to make a decision as to which ticket to recommend to the user for purchasing and may even be able to purchase the ticket. It has been reported that mobile agents generally lend themselves nicely to searches and computational tasks that require parallel processing [3, 4].

The full-scale deployment of this programming paradigm is subject to numerous security threats that are a result of the agent’s ability to migrate from one host to another. A discussion of such threats and proposed countermeasures is provided in the next section, 2.1.1

2.1.1 Mobile Agent Security

The mobility of agents may depend on a predetermined itinerary or intermediate computation results. Along with flexibility in system design, agent mobility also introduces security concerns. The security requirements of agent systems are identical to that of traditional computing environments [1]; and are classified as confidentiality, integrity, availability, authentication and non-repudiation. Confidentiality deals with ensuring the protection of information against the possibility of being disclosed to unauthorized parties. Integrity ensures that third parties cannot modify relayed information, if any such modification would be undetectable. Availability requires that attacks do not prevent information and system resources from performing their intended purposes. Authentication is concerned with ensuring that the identity of any entity in the system has been verified. Lastly, non-repudiation is intended to prevent any party from denying having performed an action, by providing the mechanisms to prove that such actions have indeed originated from the specified sender.
The violation of any of the security requirements of agent systems constitutes a threat to the security of the system. It has been noted that security threats to the mobile agent paradigm stem from insecure networks, malicious agents, malicious hosts, or any malicious entities with access to the network [1, 2, 5-8]. Using the term agent platform to refer to the agent’s host or execution environment; the security threats to an agent platform have been categorized into four main categories based on the components of the agent system [5]:

- Agent-to-Platform
- Agent-to-Agent
- Platform-to-Agent
- Other-to-Agent Platform

Agent-to-Platform threats encompasses issues arising from an agent violating the security requirements of the executing environment through masquerading, denial of service, or unauthorized access to system resources. Agent-to-Agent threats stems from violations of an agent’s security requirement by another agent in order to exploit any security weaknesses. Agent-to-Agent threats can occur through denial of service, masquerading, repudiation, or through unauthorized access if the platform has weak control mechanisms in place. Platform-to-Agent refers to instances where the platform attacks the agents through masquerading, denial of service, eavesdropping, alteration of code, or data to cite a few. Lastly, Other-to-Agent Platform is concerned with instances where the platform’s security is compromised by entities external to the agent system. Such threats can occur through masquerading, denial of service, and unauthorized access.

Proposals to secure agent systems have focused on protecting either the hosts or the agents. The security requirements of the two entities are not complementary; as the mobile agent may require anonymity, which may conflict with the requirements of hosts [2]. The execution environments of hosts provide the basic mechanisms for reception and sending of mobile agents;
this is generally achieved through interpreters. The use of interpreters serves a two-fold purpose of providing support for mobile code portability and that of executing mobile agents in a sandbox for security purposes [2]. The use of interpreted script or programming language can allow host to deny execution of potentially harmful commands [1]. Protection of hosts is also achieved through path histories and code signing to verify authenticity and source of the mobile code. The latter is instrumental in satisfying the host’s security needs for authentication, and access control.

Protecting agents from malicious hosts involves protecting their data along with the privacy and integrity of the agent’s execution, which encompasses the agent’s code and state [5]. Bierman et al. [7] have classified proposals put forth to address the issues of securing agent entities into four categories namely, trust-based computing, recording and tracking, cryptographic techniques, and time techniques.

Within trust-based computing, a host is considered trustworthy if it adheres to its published security policy; protection of the agent is achieved through provision of tamper resistant hardware, and trusted execution environments, which restricts the hosts to which an agent can travel.

Recording and Tracking an agent’s itinerary represents the second category and relies on mechanisms such as anonymous itinerary, server replication, or path histories to protect the agent. Path histories refer to the maintenance of a record of all platforms visited by the agents. Within the implementation of path histories, each host adds a signed entry to the record containing its identification along with that of the next host to be visited by the agent [5, 7]. Server replication is a mechanism that allows detection of tampering by executing multiple copies of an agent on various execution environments [1, 5, 7].

Cryptographic techniques rely on encryption/decryption algorithms to address various threats. Cryptographic tracing, and partial result encapsulation represent two of the mechanisms that fall under this category. Cryptographic tracing occurs through the generation of a signed
execution log of the agent on a host [7]. The current host passes the signed log on to the next host in the agent’s itinerary, and maintains a copy locally for verification in the future by the agent’s owner. Partial result encapsulation encrypts the result of the agent’s execution on each host using the owner’s public key [1, 5]. The incrementally encrypted data can later be retrieved using the owner’s private key.

Time Techniques protect agents by restricting the time an agent spends on any particular host to prevent evaluation or reverse engineering of the agent by a malicious host. It is worth noting that restricting the execution time of an agent may place unrealistic constraints on some agent applications. Table 2-1 provides the summary of a subset of counter-measures that have been proposed to address agent security. Despite the threats plaguing the paradigm, numerous platforms have been released to support agent-based applications; a discussion of such platforms follows in Section 2.1.2.

Table 2-1: Counter-Measures to deter security threats to Agent systems.

<table>
<thead>
<tr>
<th>Origin of Threats</th>
<th>Proposed Counter-Measures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agents</td>
<td>• Code Signing</td>
</tr>
<tr>
<td></td>
<td>• Interpreters</td>
</tr>
<tr>
<td></td>
<td>• Authentication</td>
</tr>
<tr>
<td></td>
<td>• Access Control</td>
</tr>
<tr>
<td>Platforms</td>
<td>• Time techniques</td>
</tr>
<tr>
<td></td>
<td>• Partial Result Encapsulation</td>
</tr>
<tr>
<td></td>
<td>• Cryptographic Tracing</td>
</tr>
<tr>
<td></td>
<td>• Anonymous Itinerary</td>
</tr>
<tr>
<td></td>
<td>• Server Replication</td>
</tr>
<tr>
<td></td>
<td>• Path Histories</td>
</tr>
</tbody>
</table>
2.1.2 Mobile Agent Platforms

As mentioned earlier (Section 2.1.1), the execution environment of agents is generally achieved through the use of interpreted programming languages or scripts to provide code portability. Available agent platforms have been implemented through the use of Scheme and, Tcl as well as Java; the latter representing the dominant approach [9, 10]. Altmann et. al ranked the Java-based mobile agent platforms based on Security, Availability, Environment, Development and Characteristic Properties [10]. The security criterion evaluated platforms based on support for encryption and provision of a secure execution environment; the availability parameter refers to the ease of acquiring and using the platform. The environment evaluation criterion evaluates platforms based on supported operating systems and available documentation; while the development criterion focuses on rating efficiency in designing, implementing and deploying agent applications on the platform. Lastly, the characteristic properties of the platforms are measured based on support for mobility of agents and adherence to standards of the Foundation for Intelligent Physical Agents (FIPA) [11] and Object Management Group’s MASIF [12]. Altman’s study concluded that Grasshopper, Jumping Beans, and Aglets represent the top three Java-based agent platforms, respectively.

Grasshopper [13], was initially developed by GMD FOKUS and IKV++; it integrates traditional the client/server and mobile agent paradigms. Grasshopper is conformant to both the FIPA standards and MASIF; furthermore it provides support for Secure Sockets Layer (SSL) [14] and X.509 Certificates. Jumping Beans [15], while not a mobile agent system per se, provides the framework to build an agent system by allowing applications to "jump" between hosts on a network. The framework automatically encapsulates the code and data of jumping applications in order to bypass issues relating to software/tools requirements on the receiving host. Lastly, Aglet, initially released by IBM to support the development of mobile code [16, 17], is currently
available as an open-source project. Aglets run on the Tahiti Server within the Aglets' Context, which is responsible for enforcing the security restrictions of the mobile code. The term Aglet is used interchangeably in the literature to refer to each individual mobile agent as well as the platform. Within this dissertation, the term Aglet will be followed by the term platform when referencing the actual mobile agent platform; the term will otherwise be a reference to individual agents.

The use of the Java programming language provides platforms with the ability to secure hosts through sandboxing; however, the security of the host is only as effective as the security policies put in place. Furthermore, hosts are still susceptible to denial of service attacks from agents unless processor, memory, and external resources allocated to any migrating agent are limited [1].

2.2 Sensor Network

The monitoring of an event in a particular environment in order to facilitate reaction to its occurrences is typically achieved through sensor networks. A Sensor Network is the resulting network that emerges from the possibly random deployment of multiple devices equipped with sensing apparatus, in a particular area, to perform a task through coordination and communication. The devices, and their sensing apparatus, are typically referred to as sensor nodes within the sensor network context. A sensor network is generally composed of four basic components [18], namely:

• The sensor nodes, which are equipped with one or more sensing apparatus such as seismic, heat, motion, infrared sensors to cite a few.

• A networking infrastructure, which is generally based on wireless technologies although this is not a requirement.
• A sink or base station to which collected information is relayed.
• Computing resources at the sink, or beyond, in order to perform data mining and correlation.

The nodes in the network are generally comprised of a transceiver, a memory unit, and an embedded processor for local processing. Nodes in a sensor network are, without regards to exceptions, low-power devices with memory capacity on the order of Kilobytes, and highly constrained computational power. They may be inexpensive, and may not need to be retrieved and reused should they fail. Furthermore, the nodes may be mobile if mounted on a robot. On the other hand, the base station is usually assumed to not suffer from the resource scarcity, if any, present in the other nodes and is not necessarily equipped with any sensing apparatus.

Using the sensing apparatus of the sensor nodes, the network can monitor the area, which it covers, and react to events of interest. The task is accomplished through relaying of the information of interest from the nodes to the sink for processing. Note that the transfer of information can be initiated from the sink or from the sensor nodes depending on the implementation of the network and the task at hand. The network may be composed of thousands of nodes that have been programmed before being deployed in the area of interest. The deployment of the nodes may be random, or the nodes may be placed in specific points of interest, depending upon the application at hand. A discussion of the general applications of sensor networks is provided in Section 2.2.1

2.2.1 Sensor Network Applications

Applications that need to react to events by either taking a defined set of actions or simply collecting data in an environment of choice typically achieve their goals through the deployment of sensor nodes to form a Sensor Network dedicated to the task at hand. The applications of Sensor Networks are generally classified into several main categories [18], as
introduced in the following subsections:

2.2.1.1 *Military Applications*

Sensor network in military applications address issues of monitoring friendly or enemy forces, assessing damages on a battlefield, along with biological or chemical attack detection amongst other uses.

2.2.1.2 *Environmental Monitoring*

Within environmental monitoring, the aim is to detect environmental incidents such as flood, fire, seismic activities, or biological events in the area of interest. The use of sensor networks can also be used to help large farms deal with irrigation issues, track animals on the farm or monitor the temperature in a barn.

2.2.1.3 *Home and Office Applications*

Home and office applications can also benefit from sensor network due to the sensing nodes ubiquitous nature allowing for them to be integrated in household appliances to respond to environmental stimuli or to a user’s command from a remote location, possibly over the Internet.

2.2.1.4 *Habitat Monitoring*

Within habitat monitoring, nodes in the Sensor Network are used to observe breeding patterns of wild animals and the lifecycle of plants without disturbing the environment.
2.2.1.5 Health Monitoring Applications

Health monitoring can depend on sensor nodes to carry a patient’s vital information in order to reduce errors; they can also be used to monitor a patient and react to physical events or to the patient’s vital signs.

2.2.1.6 Others

The compendium of sensor network applications is not solely represented in those aforementioned categories. There are numerous other applications of Sensor Network that do not fit into any of the aforementioned categories, as is the case for the possible use of sensor nodes to detect suspicious individuals or survivors in a disaster, interact with humans in a classroom setting, as well as track a moving object meeting some specifications in an environment. The latter represents one of the key concepts dealt with in this research, and is thus introduced in more detail in the following section, namely 2.3

2.3 Target Tracking in Sensor Networks

Applications of Sensor Networks have to deal with the inherent limitations that exist in the environment; such limitations include the scarcity of the power available in the nodes. The use of Sensor Networks in numerous applications, such as target tracking, has been the focus of numerous research efforts (see Section 2.2).

Target Tracking deals with the issue of following a particular object as it moves within an environment. Within the scope of Sensor Networks, the environment is limited to any area where the nodes involved in the tracking are present. Numerous schemes have been put forth to allow sensor nodes to track a target efficiently while minimizing the power consumption of the network
as a whole. Pattem et al [19] have identified four major approaches used in Sensor Networks to track a target. The classification of the approaches is based upon the scheme used to minimize power usage in the network in tracking a target. The first category: Naive Activation (NA) refers to the scheme in which all the nodes in the network are activated at once to track a target. The second approach, Randomized Activation (RA), activates node based on a probability. Selective Activation based on Prediction (SA) is the third strategy, which tries to predict the future location of the target and only activates the nodes whose range the predicted location will fall within. The last category is named the Duty-cycled Activation (DA), which is characterized by the fact that the entire network is turned on and off periodically. A summary of the activation methods along with their characteristics, herein discussed, is provided in table 2-2.

Table 2-1: Activation Methods and Associated Characteristics.

<table>
<thead>
<tr>
<th>Activation Method</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naïve</td>
<td>Activates all nodes in the network simultaneously</td>
</tr>
<tr>
<td>Randomized</td>
<td>Activates nodes based on probabilistic estimation</td>
</tr>
<tr>
<td></td>
<td>that target is in the range of a node</td>
</tr>
<tr>
<td>Selective</td>
<td>Activates nodes based on predicted path of the target</td>
</tr>
<tr>
<td>Duty-Cycled</td>
<td>Periodically activate and de-activate the entire network</td>
</tr>
</tbody>
</table>

The effectiveness of the Randomized Activation as well as the Selective Activation based on Prediction schemes rely heavily on the accuracy of predicting the future location of the target being tracked. In order to deal with the randomness of a target’s trajectory, Yang et al [20] introduced the Distributed Predictive Tracking (DPT) algorithm, which views the sensor network as a clustered system, with each cluster having a cluster head along with border sensors that are always active. To track an object, the cluster head activates three sensors that can detect the target using their low sensing beam; if three such sensors cannot be found, the cluster head chooses three sensors that can detect the target using their high beam. The cluster head determines the
next location of the target and informs the next appropriate cluster head of the ID of the tracked target. The system recovers from failure by periodically sensing the range of the three nodes involved in the tracking from low beam to high beam; if the lost target is still undetectable, sensors within a certain radius are increasingly activated. Mechitov et al [21] modeled the motion of the target through piecewise linear approximation and the introduction of a distributed algorithm, named Cooperative Tracking, to help in the prediction of the targets’ future location. The distributed tracking algorithm is based on four steps; during the first step, each node measures the duration for which the tracked object is within its range. In the second step, the nodes exchange the recorded values for use in the third step to estimate the object’s location by computing the weighted average of the detected values. Step four runs a line-fitting algorithm on the set of points calculated in the third step.

Furthering the trend of taking a distributed approach to tracking a target in Sensor Network, Zhang et al [22, 23] introduced the Dynamic Convoy Tree-Based Collaboration (DCTC) algorithm. DCTC can be used to track moving targets in sensor network focusing on the issue of collecting and generating reports as the target is tracked. The proposed algorithm relies on nodes around the target collaborating to determine a root, the characteristics of a suitable node were left as implementation detail; it has been suggested however that the node closest to the target would be a fair candidate to act as root. Once a root has been determined, it is responsible for predicting the path of the target and activating nodes along that path. Limiting the environment to binary sensors, Aslam et al [24] dealt with the issue of approximating a target’s location by requiring nodes to relay to the sink one bit of information dictating whether the object of interest is moving closer or away from the node. The approach minimizes communication between nodes and the sink while allowing for the target’s trajectory to be predicted by the sink. Taking an information-driven approach, Zhao et al [25] proposed to inject queries into the area where the target is likely to appear initially. Once the query has reached a node where the event is likely to occur, the node
upon detecting the event, determines the next best sensor to which to pass the information for tracking the event based on not only an estimate of future direction of the object but also on a minimized communication cost to transfer the information of the target to the next node. Distributed approaches to target tracking have eventually led to the use of mobile agents to perform the task. This is evidenced by one of the experiments conducted in the UC Davis mobile agent platform for Sensor Networks [26] introduced within the scope of reconfigurable sensor network. Further details on reconfigurable sensor network is provided in the following Section, namely 2.4

### 2.4 Dynamic Tasking in Sensor Networks

Reconfigurable sensor networks have been introduced in order to support dynamic tasking of sensor nodes and to allow for the network to concurrently support multiple applications. One approach to achieving a reconfigurable sensor network is to consider the sensor nodes as a set of data-stores into which queries can be injected to collect information that can be used by the sink for a given purpose. Collecting information from the sensor nodes is inadequate in applications where the nodes need to interact with each other in order to reach a conclusion in real-time [27], as would be the case in distributed target tracking applications or any applications that require the use of distributed algorithms. The notion of “active sensors” was introduced in [27] to denote the path to addressing the reconfigurable need of sensor nodes through abstraction of the runtime environment. A recent trend in rendering nodes reprogrammable focuses on the hardware and exploit the benefits offered by Field Programmable Gate Arrays (FPGAs) to dynamically reconfigure the hardware. We present in this section a discussion of the proposed work that has been conducted along the path of reconfigurable nodes in the research community, covering active sensors (Section 2.4.1) and FPGA-based sensor nodes (Section 2.4.2).
2.4.1 Active Sensors

The “active sensors” approach typically makes use of virtual machines, script interpreters and mobile agents to render sensor nodes reprogrammable. We present in this section a discussion of the proposed work that has been conducted along that path in the research community. Section 2.4.1.1 introduces Mate [28, 29], followed by SensorWare [27] in Section 2.4.1.2, Deluge [30, 31] in Section 2.4.1.3, and Agilla [32] in Section 2.4.1.4. Section 2.4.1.5 introduces a mobile agent framework developed at UC Davis [26]; finally, ActorNet [33] is discussed in Section 2.4.1.6; while Section 2.4.1.7 introduces SOS [34]. In general, the systems included in our discussion have targeted their developments to suit applications requiring low-cost reprogrammable nodes with no restrictions on the maximum physical size of the node. With the exception of SensorWare, the storage requirements of these systems are such that they can inhabit the Berkeley Mica motes having a 4Mhz microprocessor and 136KB total memory (Flash, SRAM, EEPROM). The application domain of these systems range from military to environmental and habitat monitoring; it does not however span over to applications requiring particularly small sensors or those that do not benefit from reprogrammable nodes.

2.4.1.1 Mate

Mate aims at providing sensor networks with a flexible architecture upon which application specific scripting environments can be built [28, 29]. Mate consists of three major components, which are contexts (units of concurrent execution), operations (units of execution functionality), and capsules (units of code propagation). A Mate virtual machine (VM) component can be either part of the basic template, which is general or a part of the specific VM tailored to an application domain. The basic template of Mate includes the scheduler, the
concurrency manager as well as the capsule store. The basic template does not handle data storage but instead defines a set of types used for transferring data to and from functions. The specific VM of Mate focuses on mapping contexts and operations to the basic template. The set of operations consists of the instruction set of the specific VM and the contexts trigger the execution of the VM.

The scheduler represents the core of Mate and operates based on a FIFO algorithm. The scheduler interleaves the execution of contexts from a queue of executable ones through the concurrency manager. The concurrency manager operates upon the assumption that sensor networks have few resources. Hence before running a context, the concurrency manager checks to see if all the resources that the context might need are available; if so it submits the context to the scheduler, else the context is added to a wait queue until they are eligible for execution. The capsule store handles capsule loading and storage. In the original version of Mate, the capsule store used broadcasting to propagate code over the network and reprogram nodes. The approach was not efficient and had the drawbacks of having nodes broadcasting fragments of code repeatedly even though the whole network has already been reprogrammed. In the new version, Mate makes use of Trickle [35], a protocol designed to address the issue of code maintainability in sensor network, to update the network. Trickle’s main contribution is in limiting broadcasting between neighboring nodes through suppression and dynamic adjustment of the broadcast rate. Nodes in Trickle remain quiet until they hear a summary of the code to be disseminated that is older than the one being maintained locally. In such cases, Trickle nodes broadcast the maintained summary to update the other nodes. If a summary identical to the maintained one is heard over the broadcast channel, the node remains quiet.

With the help of Trickle, Mate provides an effective approach to sensor network programming through application specific virtual machines. User programs in the environment are generally short and simple as the VM provides the common functionalities to specific
application domains. The current framework, due to its independent triple layer approach, can be optimized at any layer to improve performance either based on power usage or reduced cost of propagation and execution. This is an interesting approach to address the issue of reconfigurable sensor networks; however Mate suffers from assumption that the reprogramming occurs over all the nodes in the network and they assume that all the nodes are coordinating for the execution of one specific application at any one point in time. Furthermore, Mate views the network as an isolated entity and does not address issues of interoperability with other networks.

2.4.1.2 SensorWare

SensorWare, introduced as an attempt to address the issue of reconfigurable sensor network, runs on top of an Operating System (OS), which handles the standard functions and services of a multi-threaded environment [27]. The OS in turns constitutes a layer above the Hardware abstraction layer running on top of the Hardware layer. The OS, along with the SensorWare layer, can also be accessed by the Application Services, which may be distributed but not mobile, such as location discovery. The Application Services layer is at the same level as the Scripts, however the Scripts only interact with the SensorWare layer and may be mobile.

The SensorWare layer is comprised of the language as well as the run-time environment for the mobile scripts in the network. The SensorWare scripting language is based on the widely popular scripting language Tcl, augmented with some functionalities that are suitable for sensor network environment. The SensorWare language is event-based and can be thought of as a state machine influenced by external events. Each event is tied to a specific handler that executes when the event occurs. An event may fire one or more events or change the state of the system while it executes. A SensorWare script waits on events and calls the appropriate handler when an event occurs; the script can then wait on a new set of events or loop around and wait on the same set of
events after the execution of the handler.

The run-time environment of SensorWare consists of three main components: the Script Manager, the Admission Control Task and Policing of Resource Usage Task (ACT-PRUT), and the Resource Handling tasks. The ScriptManager as the name suggests manages the scripts; including their creation, the record keeping of their data as long as the script is alive. The ScriptManager also keeps a copy of the script’s code in the cache to reduce transmission. The ScriptManager only create scripts if it receives a positive reply from the ACT-PRUT component of SensorWare, which ensures that scripts are not over-consuming the resources. The ACT-PRUT also monitors the energy level of the system; in case it is low, the ACT-PRUT decides which scripts should be suspended to allow the maximum number of scripts to execute to completion.

The Resource Abstraction and Resource Metering tasks or Resource Handling tasks are a set of task managing specific resources depending on the node. The Radio/Networking and Timer Service are the only two standard tasks available on every node. The former manages the radio by accepting network messages, requests about the format of network messages, and measures radio utilization of scripts. The latter processes timer requests from scripts using a real-time timer.

Furthermore, SensorWare deals with the issue of sensor nodes having different sensing capabilities by adopting the notion of virtual devices from Linux.

SensorWare provides the necessary support allowing a Sensor Network to run multiple scripts simultaneously; as such, unlike Mate, it does not assume that the whole network is focused on only one task at any point in time. On the other hand, just as Mate, it ignores issues of interoperability. The latest implementation of the system required 179KB of space with the core accounting for 30KB, which makes it very unsuitable for environments populated with nodes having very few storage capabilities.
2.4.1.3 Deluge

Deluge has been designed to handle the dissemination of large data objects over Wireless Sensor Network (WSN) [30, 31]. Deluge is aware of the network density, and is built to handle the unpredictability of nodes availability by representing data objects as a set of fixed-sized pages, which allows for multiplexing and incremental upgrades. Deluge, just as Mate, is based on Trickle. Trickle focuses on single packet dissemination, while Deluge addresses the multiple packets aspect. Deluge ensures transmission reliability by incorporating work in reliable data transfer protocols such as Pump Slowly Fetch Quickly (PSFQ) [36], Reliable Multi-Segment Transport (RMST) [37] as well as Multi-hop Over-the-Air Programming (MOAP) [38].

Deluge represents data objects in terms of pages, a fixed number of packets; and divides objects into multiple pages. It requires that a node function in one of three states, which are: Maintain, RX and TX. Deluge defines a page to be complete if every packet for that page has been correctly received; moreover a page is available if it is complete and every preceding page is also complete. In the maintenance state, a node is responsible for all nodes within range having the newest version of the object profile and all data for the new version. This is accomplished through advertisement of the node’s object profile; redundant messages are eliminated by the use of Trickle. A node in maintenance mode transitions to the RX state if it receives an advertisement with the same version number as its object profile and with more pages available; unless it has recently received a request for one of its pages or it has recently received a packet to update one of its currently complete pages or the next page in the sequence. A node transitions to state TX once it receives a request for one of its available pages in the current version. A node in the RX state actively requests the packets needed to complete a page p>y where y is the last complete page in the sequence. Deluge makes use of selective negative acknowledgement (SNACK) [39] and broadcasts a bit-vector specifying the packets in a page that are still needed. Each node
uses a random back off to minimize collisions; the back off is also used to increase efficiency as if one of the needed packets is overheard on the channel during the back off period, it can be intercepted and does not have to be requested again.

The Deluge protocol tackles the issue of network programming in sensor networks very efficiently. It takes into account the density of the network as well as the connectivity issues that exist in the environment. However the new protocol is limited by the fact that it currently assumes that all the nodes in the network need to be programmed and thus focused on propagating code updates to all nodes in the network. It is conceivable that an administrator may only want to update particular nodes in the network. Furthermore, it ignores issues of interoperability, and the need to support multiple tasks.

2.4.1.4 Agilla

Providing support for reconfigurable sensor network has served as the main motivation for the development of Agilla [32]. Agilla allows each node to support multiple agents that may or may not be cooperating to accomplish a task. Agilla allows for agents communication through tuple spaces, which are nothing more but an ordered set of fields of types and values. Agents in Agilla can react to changes in tuples if they notify the Engine that they are interested in a specific tuple template. Agilla supports four local atomic tuple space operations and four remote tuple space operations, which are non-blocking. One assumption made by Agilla is that each node knows its geographical location, which is used as the address of the node. Agents in Agilla can clone themselves or move to another location carrying with them either their code and state or just their code. Agilla agents die at the completion of their task to allow for efficient memory usage.

Agilla's architecture consists of three main layers: the mobile agents layer, the Agilla
layer, and the TinyOS layer. The Agilla layer is further subdivided into five major components; an Agent Manager, a Context Manager, an Instruction Manager, a Tuple Space Manager, and an Agilla Engine. The Agent Manager handles memory allocation for agents as well as notifying the Agilla Engine when an agent is ready to run. The Context Manager handles context information for a node such as neighbor-list and the location of the node; it is responsible for the discovery of neighbors through the use of beacons. The Instruction Manager is used to overcome the fact that TinyOS [40] does not support dynamic memory allocation. When agents arrive on a node, they specify the amount of instruction memory that is needed; the Instruction Manager handles the memory allocations along with the sequence of instructions to be executed. The Tuple Space Manager controls memory allocations for tuples. It implements the non-blocking tuple space operations and manages the reaction registry, which keeps track of agents and their tuples of interest. Once an inserted tuple matches the template of interest of one agent, the Tuple Space Manager notifies the Agent Manager, which then execute the reaction code of the agent. The Tuple Space Manager handles packaging and restoring an agent’s reactions as it moves through the system. The Agilla Engine handles the scheduling of Agents for execution in a round-robin fashion. It is also responsible for sending and receiving agents between nodes and can be considered to be a virtual kernel controlling the current execution of Agents on a node.

Agilla has been designed to allow more than one agent to execute on a node. However, similarly to the previously discussed works, Agilla does not attempt to allow different networks to collaborate, nor are services provided to migrating agents.

2.4.1.5 UC Davis Mobile Agent Framework

Researchers at UC Davis have put forth a mobile agent framework built on top of the Mate virtual machine to allow the deployment of the agent-programming paradigm within sensor
network environment [26]. The framework allows agents to execute within an interpreter that attempts to prevent node crashing from corrupted agents. The interpreter implements the basic functionalities of agents, such as agent forwarding, so as to minimize the size of agent code that needs to be transferred from node to node. The interpreter supports agent migration through both unicasting and broadcasting communication models, the decision as to which model to be used is left to the agent so as to increase efficiency based on the needs of an application. Agents can also decide on whether or not to request acknowledgements during migration depending upon their fault tolerant requirements. Agent communication is achieved through "bread crumbs", inspired by Ant Colony Systems (ACS), by writing its state on a node or by reading stored states, which are preserved for the duration of the node’s lifetime. A case study for the use of agent in sensor network is conducted leading to the conclusion that the use of agents is particularly beneficial in the environment where only a subset of the network needs to be reprogrammed or the frequency at which nodes are reprogrammed increases.

The mobile agent framework for Sensor Network introduced by researchers at UC Davis suffers from the same limitations as Mate, including the fact that cross-network interoperability is not addressed, with the exception that the entire network does not have to be reprogrammed. As an agent system, similarly to Agilla, interoperability is possible under the assumption that the different networks support the same agent platform and the same set of protocols.

2.4.1.6 ActorNet

ActorNet [33] is a mobile agent system for Wireless Sensor Networks, which supports an asynchronous communication model, context-switching, multi-tasking, agent coordination as well as virtual memory. The agent system can be thought of as two entities: the agent language and the platform design. The Actor language is based on Scheme and provides users with the basic
functionalities such as the ability to send and receive messages. Other agent primitives can be implemented using the language. The language allows agents to migrate with their states by considering the state of the agent to be a pair of continuation, which can be obtained by an operator call, innate to Scheme, and represent the remaining portion of the program to be executed, and a value to be passed to the continuation. The ActorNet platform is a virtual machine that can support multiple actors (agents) per node. The platform provides a unified environment for actors, which interact exclusively with the interpreter at the top level of ActorNet's architecture. The platform was designed for the Mica2 motes of Berkeley running the TinyOS operating system.

ActorNet took a very efficient approach to the issue of reconfigurable sensor nodes, as did Agilla and the researchers at UC Davis, through the adoption of mobile agents to reprogram nodes. The platform, however, does not allow interoperability of heterogeneous networks.

2.4.1.7 Reconfigurable Sensor Network with SOS

SOS is a sensor network operating system supporting run-time reconfiguration of software [34]. The development of SOS was motivated by the fact that physical reprogramming of thousands of nodes is a wasteful activity in terms of time and resources since the nodes are generally deployed in hazardous, remote and fragile fields. Furthermore, the introduction of SOS is meant to allow the updates of modules without interrupting sensor operation while providing the flexibility of virtual machines without the associated cost of interpreted languages. SOS is composed of a statically compiled kernel providing system services to dynamically loadable binary modules implementing drivers, user programs etc... The system core implements basic services such as asynchronous message, low-layer hardware abstractions and fixed-partition dynamic memory. The modules in the system communicate through message passing and access
services through a jump-table in memory. While providing services and allowing for a node to be dynamically tasked, SOS still views sensor networks as isolated entities. Furthermore, the nodes in the network do not perform processing of acquired data at the point of collection, instead opting for the relay of such data to a base station.

2.4.2 FPGA-based Sensor Nodes

The recent approach to render nodes reconfigurable through the use of FPGA has been highly motivated by the need to increase the computational power of nodes in order to allow some local processing of data. The Virtual Architecture for Partially Reconfigurable Embedded Systems (VAPRES) has been put forth to that end based on the observation that FPGA can outperform typical microprocessors found in sensor nodes [41]. The introduction of VAPRES is also motivated by the inability of Agilla (see Section 2.4.1.4) to handle video feeds and other advanced sensor data. Using VAPRES, advanced sensor data can be processed without halting execution of the device. The proposed architecture relies on the ability of some FPGAs to be partially reconfigured by modules in order to react to environmental observations. VAPRES handles inter-module communication and consists of a central controlling agent, a flash controller core to read and store partial bitstreams along with other peripherals for communication. In order to track a target, VAPRES modules can be dynamically loaded and unloaded into Partially Reconfigurable Regions (PRR) based upon three possible scenarios:

- A tracking module processes new input, loads the appropriate specialized module then waits for new targets.
- The central controlling agent successively loads all tracking module into a single PRR and allows processing of data for a short period time. The module with the best result is then reloaded and assigned tracking responsibility of the target.
• The central controlling agent loads all possible modules in parallel to track the target for a short period of time and then allows the module with greatest accuracy to remain loaded.

The authors envision the ability of FPGA to meet the growing requirement for sensor networks to lower power consumption while increasing processing power. The reconfigurable methodology introduced improved processing time while reducing power consumption by 5% to 25%. VAPRES approach, while efficient, considers the node as an isolated entity and thus does not address issues of interoperation with existing networks. As thus, adoption of VAPRES in large deployments will be deterred, since the network will remain specialized based on its localized sensing ability.

Commuri et al. motivated by the need to provide in-network data aggregation also adopted the notion of FPGA-based nodes [42]. As per their proposal, Reconfigurable Cluster Heads (RCH) are used to aggregate data from other nodes in the network and relay it to the base station for actual processing. The election of RCHs is done based on the energy available from participating node, with the RCH being the node with the most energy. The RCHs are essentially a set of distributed aggregation centers, which can be queried for data. The reconfiguration of RCHs is query-based, in that RCHs are reconfigured based on specific aggregation algorithm of incoming queries. This represents a considerable drawback in the proposed work, as the rate of arriving queries and their heterogeneity may require a drastic number of reconfigurations to be performed. While the proposed approach is limited since it does not take advantage of the power of FPGAs to process the data close to the point of collection; it does however allow for the possible bridging of sensor networks with established infrastructure through the RCHs hence enabling the foundation for an interoperable system. Table 2-3 provides a comparative summary of the systems discussed in this section covering active sensors and FPGA-based ones. As interoperation of sensor networks is one of the core components of this dissertation, it is discussed in more details in the following section.
Table 2-3: Comparison of Approaches to Sensor Network Re-Configurability.

<table>
<thead>
<tr>
<th>System</th>
<th>Platform</th>
<th>Heterogeneous Tasks</th>
<th>Interoperability of networks</th>
<th>Service-Oriented Infrastructure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mate</td>
<td>Virtual Machine</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SensorWare</td>
<td>Run-Time Environment</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Deluge</td>
<td>Network Programming</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Agilla</td>
<td>Agent System</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>UC Davis framework</td>
<td>Agent System</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>ActorNet</td>
<td>Agent System</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SOS</td>
<td>Binary Modules</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>VAPRES</td>
<td>FPGA-based</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>RCH</td>
<td>FPGA-based</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

2.5 Collaboration in Sensor Network

Recent advances in sensor networks have focused on the need for the network to evolve and handle complex dynamic tasks that may require data from multiple existing networks. The ability to harness data from neighboring or geographically overlapping sensor network represents a stepping-stone to accomplish such feat. The interoperability of Sensor Networks with Enterprise Networks motivated the introduction of Edge Servers by Rooney et al. [43]. The Edge Servers are responsible for the aggregation and filtering of collected data to be relayed to the Enterprise Network in order to reduce the system’s load. Edge Servers are externally powered sensors with adequate processing capability and TCP/IP support. Edge Servers comprise multiple input channels and one output channel. The channels support asynchronous communication through message-bins with an associated controller regulating the topics available on the channel. Furthermore, the channels announce their topics and other pertinent information over a meta-channel supporting multicast semantics. Using boosters, Edge Servers asynchronously process,
aggregate and filter data from their input channels prior to publishing the result onto their output channel. The Messo protocol was introduced [43] to enable data-centric sensors to publish into a channel and communicate with the network. While the proposed approach is promising, it is narrow as it focuses on enabling interoperability between Sensor Network and Enterprise Networks. The proposal is also limited by the fact that the system does not process data near the point of collection focusing instead on relaying it to high-powered nodes for processing.

Envisioning the existence of heterogeneous sensor networks, with possibly mobile nodes, Tilak et al. [44] studied the issue of Dynamic Resource Discovery (DRD). The work divided resource discovery in sensor network into the tasks of identifying what resources need to be tracked and that of querying the network in an energy-efficient manner. Using metadata collection, identifying what resources need to be tracked is resolved as the metadata provides information regarding communication protocol, format of messages and other pertinent information for interoperability. The granularity of attributes to be tracked is however left to application developers. Cluster heads are used to hold resource attributes and respond to queries, allowing other sensors to conserve energy. The cluster heads in the system are used in a data-centric aspect, as they are solely responsible for relaying data to a remote sink. Hence, data is still processed remotely by a base station. The work does not however venture into how various networks communicate in exchanging metadata.

Enabling the sharing of physical resources by a dynamic subset of sensor nodes led to the introduction of Virtual Sensor Networks (VSN) [45]. VSNs are formed using subsets of one or more physical sensor network nodes in order to accomplish a common task. As a VSN may not be fully interconnected nor connected to a base station, the physical nodes not comprising the VSN are used to provide communication support to the VSN. VSN perform two major functions, which are VSN maintenance and membership maintenance. The former deals with addition and deletion of nodes; joining or splitting VSNs as well as deriving boundaries. The latter manages
the roles of sensors in the VSN for communication or processing. The use of VSN is applicable in areas where geographically overlapping sensor networks are deployed. However, data from the VSN is still relayed to a base station for processing and there exists the underlying assumption in the proposal that cooperating networks use the same communication medium and thus can easily exchange messages.

Tiny Web aims at providing interoperability through adoption of the notion of web services to the sensor network environment [46]. Tiny Web addresses interoperability at both the network and application layers of communication. Using Web Service Description Language (WSDL), nodes in Tiny Web advertise their interfaces to applications. Under the assumption that only simple responses specified by the nodes are expected and that messages comply to WSDL, code complexity and data overheads are reduced. Furthermore, the use of Web Services Eventing allows node to advertise their key methods as web service events in order to support the “sleep” requirements of nodes. Tiny Web relies on a standardized mean of communication between networks and focuses on dealing with data representation in order to provide interoperability.

Semantic Sensor Web (SSW) [47] enables interoperability through the use of metadata and contextual information from networks. SSW proposes the use of semantic metadata and contextual information to foster the use of sensor data from multiple sources. It thus addresses interoperability by focusing on the ontology/semantics of data to capture their formal definition and answer complex queries. SSW builds an ontology-based hierarchical system allowing access to appropriate sensed data through web applications. As annotation and query processing is performed by powerful nodes, such as base stations, sensed data still needs to be relayed and thus not processed near their point of collection. Furthermore, communication heterogeneity is not supported and left as a detail to be dealt with by developers.

Global Sensor Networks (GSN) is a middleware introduced to facilitate query processing, dynamic processing and aggregation of sensor data [48]. No assumptions is made in GSN
regarding implementation details of the underlying networks except for the existence of a sink connected to a base computer through a GSN Wrapper. GSN relies on virtual sensors, which combines the input stream from various sensor nodes into one output data stream. Virtual Sensors are identified based on a unique naming attribute that is an integral part of the virtual sensor’s specification along with other attributes such as wrapper, metadata, SQL-based specification etc… The virtual sensors specification is relayed to interested parties to allow addressing and communication. A virtual sensor can be anything from a physical sensor to a sink or set of physical sensor nodes. GSN is a data stream system in which the streams are time-stamped tuples aimed at enabling a data-oriented “Sensor-Internet”. Processing of the input data streams is controlled by the specification attributes such as “Wrapper” which denotes the stream’s format. GSN processing is event-driven based on the time-stamped stream. The system relies on GSN containers to maintain and manage virtual sensors. Inter-GSN containers communication is achieved over the Internet and Web protocols. Using the notion of wrappers, means that GSN can support inter-operation with existing networks requiring a minor change to such networks. However since inter-network communication has to go through a base computer, it limits interoperability to a high level where each base computer becomes the communication portal to the underlying network.

IrisNet aims at providing an interface allowing users to query a vast amount of data collected over various heterogeneous networks [49]. The approach taken by IrisNet is to view WSN as entities capable of providing services to consumers. As such, the system was designed around two main entities, which are organizing agents (OA) and sensing agents (SA). OAs are responsible for collecting and organizing sensed data that will enable them to answer queries related to the service that they provide. Service developers orchestrate the deployment of a group of OAs for a particular service. SAs on the other hand are responsible for collecting raw data from sensors. SAs allow services to specify senselet used to dynamically filter the SAs’ raw data
for a particular service. The processed information from senselets are relayed to the appropriate OAs. Data is aggregated and filtered by SAs residing on base computers, thus sensors still have to relay sensed data for remote processing. Furthermore, IrisNet leaves the issue dealing with the communication heterogeneity of different networks to developers. As a recap, table 2-4 provides an evaluation summary of the herein discussed interoperable proposals.

Table 2-4: Evaluation of Interoperable WSN Proposals.

<table>
<thead>
<tr>
<th>System</th>
<th>Service-Oriented Infrastructure</th>
<th>Inter-WSN Communication</th>
<th>Shared Communication Protocol</th>
<th>Data Processing on Sensor Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tiny Web</td>
<td>Yes</td>
<td>Direct</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Virtual Sensor Network</td>
<td>No</td>
<td>Direct</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Semantic Sensor Web</td>
<td>No</td>
<td>Indirect</td>
<td>Not Required</td>
<td>No</td>
</tr>
<tr>
<td>Global Sensor Network</td>
<td>No</td>
<td>Indirect</td>
<td>Not Required</td>
<td>No</td>
</tr>
<tr>
<td>IrisNet</td>
<td>Yes</td>
<td>Indirect</td>
<td>Not Required</td>
<td>No</td>
</tr>
<tr>
<td>DRD System</td>
<td>No</td>
<td>Unspecified</td>
<td>Unspecified</td>
<td>No</td>
</tr>
<tr>
<td>Edge Servers</td>
<td>No</td>
<td>Indirect</td>
<td>Not Required</td>
<td>No</td>
</tr>
</tbody>
</table>

2.6 Summary

The core of this chapter dealt with various research fields representing the necessary information in order to gain an in-depth understanding of the remainder of this dissertation. Mobile agent technology was introduced in Section 2.1 and the security threats and proposed countermeasures were highlighted along with their limitations. A discussion on sensor networks’ architecture and its applications followed in Section 2.2. We dove into the use of sensor network for target tracking in Section 2.3 highlighting the latest proposals and their restrictions. Finally, we presented the issue of reconfigurable and interoperable sensor networks in Sections 2.4 and 2.5 respectively, along with the advantages and limitations of various proposals put forth thus far.
We will now transition towards introducing our work in agent security as laid out in the next chapter.
Chapter 3

Mobile Agent Security

It has been suggested by the Computer Security Division of the National Institute of Standards and Technology that one of the main hindrances to the adoption of mobile agent technology stems from the security concerns of hosts [1]. It is fair to note that any mobile agent system is suitable to support distributed applications; hence, securing such systems need to take into account the distributed nature of the environment. Malicious agents are not a threat solely to the current execution environment but to any host to which they may migrate (see Section 2.1). As such, collaboration can allow hosts to learn from each other’s experience in executing an agent’s code. In light of this observation, we herein introduce a boosting-based monitoring system that allows hosts to learn and classify agents, not only based on their own observations, but also based on collaboration with other hosts in the network, some of which may have been victims to agent attacks.

The issue of protecting a resource in a network is not novel, and has been explored in the literature under the banner of Intrusion Detection Systems (IDS). IDS attempt to detect intruders in a system in order to maintain the integrity of said system. Thus, the use of IDS should help in protecting hosts against misbehaving agents. However, the term intruder in IDS generally refers to the malicious activities of users, and not of code capable of replication and migration to and from outer systems. Essentially, IDS target traditional networking environment and not agent-based systems. Through our proposal of having hosts collaborate based on their experience with specific agent applications, we contend that the herein introduced work can help in protecting hosts in agent-based systems.
The work is based on the assumption that a malicious agent on one host is likely to represent a threat to future hosts that it will visit. In tackling the issue, our approach is to introduce a boosting-based distributed and adaptive security-monitoring framework. The proposed work is a collaborative trust scheme targeting agent systems in which the platform is secured through the use of security agents collaborating in the stead of hosts to learn from each other’s experience in dealing with migrating agents. Our contribution can thus be summarized as follows:

- Collaboration between hosts to identify malicious agents, and
- Ability of the security agents of hosts to learn from experience and prevent attacks.

The proposed framework, having been implemented in Aglet further required the analysis of the security mechanisms present in the Aglet server, Tahiti. The study resulted in the introduction of a new Aglet server providing:

- Secured communication,
- Controlled resource consumption of agents, and
- Integrity and reliability of agent’s data.

In further detailing our work, we will start by introducing the necessary background in the following section. The chapter will then discuss the adaptive security-monitoring framework, along with a discussion of our work in securing Tahiti. We will then proceed to evaluating the proposed framework and draw conclusions highlighting our contribution and future work.

### 3.1 Background

We have introduced issues pertaining to agent security in section 2.1; we will focus on IBM Aglets, intrusion detection and supervised learning in Subsections 3.1.1, 3.1.2 and 3.1.3
respectively. The discussion is intended as a concise reference to pertinent research topics in order to ascertain the value of our contribution to the community.

3.1.1 IBM Aglets

We have discussed the general classification of security threats to agent systems, and have introduced some of the agent platforms available to support the development of agent applications. Amongst those platforms introduced, the IBM Aglets platform is available as open source, easy to install and have been ranked as one of the top three platforms available (see Section 2.1.2). We have thus selected the Aglet platform as the implementation platform for our proposed collaborative approach to agent security.

Aglets run on the Tahiti Server within the Aglets’ Context, which is responsible for enforcing the security restrictions of the mobile code. Aglets may adopt seven different states during their life cycle [16, 17]:

- Activated: Aglets are activated when they are loaded, along with their states, from secondary storage and allowed to resume execution.
- Deactivated: deactivating an Aglet involves saving its state in secondary storage and halting its execution in the current context.
- Cloned: an Aglet clones itself by producing a copy of the Aglet, which will start executing in the current context.
- Disposed: disposing of an Aglet will cease its execution and remove the Aglet from the current context.
- Created: this is the state during which the Aglet is initialized and starts execution in the appropriate context.
• Dispatched: an Aglet enters this phase when it is removed from its current context and sent to another context that may exist on a remote host where the Aglet will resume its execution.

• Retracted: a retracted Aglet is an Aglet that has been moved from a remote context into the current context where the retraction has been initiated.

Fig. 3.1 provides a pictorial representation of the different states in an Aglet’s life cycle. The Tahiti server consists of two main layers: the Runtime Layer, and the Communication Layer on top of which it runs. The tight coordination between these two layers provides Aglets with their execution environment. The Runtime Layer manages Aglets’ byte code, and enforces the security restrictions in effect in the environment. The Communication Layer, on the other hand, provides the basic mechanisms to allow Aglet’s mobility and message passing.

Figure 3-1: Lifecycle states of Aglets

3.1.2 Intrusion Detection Systems

Intrusion Detection Systems (IDS) are tasked with detection and reporting of suspicious behavior in a network with the purpose of identifying intruders and thus securing the system. IDS techniques can be categorized based on the approaches taken to accomplish their goals, thus, we
have anomaly detection and misuse detection approaches [50-52]. Misuse detection leaves the specification of abnormalities to the administrators of the system. An exhaustive definition of such abnormalities is very difficult to achieve [50-52]. On the other hand, anomaly detection relies on the system’s training to learn the expected behaviors of users and is theoretically capable of detecting novel forms of attacks. In general, IDS are subject to false and missed alarms. False alarms refer to the detection of an intruder when the intruder in question has the proper rights to execute within the system. Missed alarms, on the other hand, refer to cases where an intruder is operating in the system and the IDS does not detect the presence of the intruder.

Dotted with the ability to learn from novel attacks, anomaly detection is an interesting approach to protecting hosts in an agent system, as precise prior knowledge of agent’s intents and tasks are not necessary. In general, current proposals in IDS, within the scope of agent security, suffer from the inability to detect precisely which agent is responsible for an attack as they are typically independent threads of execution of the hosting agent server process. Applying anomaly detection to securing agent hosts would thus require an IDS system that is intimate with the agent platform in use and able to detect anomalies at fine granularities.

3.1.3 Supervised Learning

Supervised learning is a subset of data mining that focuses on the ability to extract patterns from a set of raw observations to infer a learning function. Extraction of the patterns occurs during a training phase yielding the learning function used to classify future data presented to the supervised learning system. Various algorithms have been introduced to allow extraction of existing patterns in a data set. Such algorithms include Support Vector Machines (SVM) and decision trees, as well as boosting [53-56]. Support Vector Machines attempts to construct and maximize a separating hyper-plane upon mapping the data onto a higher dimensional space; the
separating hyper-plane is then used to classify future data samples. Decision trees generate a learning function by constructing a tree with the leaves representing the potential class labels. The branches of the decision tree rely on the observed features from the training data in order to yield any particular classification label or leaf. Within boosting, training occurs in stages. In each stage of boosting, a weak classifier is trained using a subset of the raw data. The set of trained classifiers yield the learning function used to determine how to categorize future data samples.

3.2 Addressing Mobile Agent Security through Agent Collaboration

Our approach in securing hosts relies on the ability of agents to collaborate, in the stead of hosts, to prevent violations of security policies. Agent collaboration has been the focus of various research efforts in recent years. Becker et al showed that incorrect confidence-integration might propagate in a multi-agent system by assuming that trust is not an issue [57]. Our system does take trust into account and is very flexible in integrating confidence to reach a decision. Chen et al. [58] presented a boosting-based hierarchical learning algorithm with inter-dependent classifiers, for experience classification. The system is however not suitable to address security concerns such as herein discussed as it is built upon the assumption that the agents are teammates, thus ignoring any trust issues. Furthermore, as a hierarchical system, the final decision must originate from the root of the structure thereby introducing a single point of failure, namely the root, if the system were to be used in distributed security monitoring.

The idea of hosts collaborating to detect intruders has been introduced in Intrusion Detection Systems [51]. The work referenced in [51] aims at reducing the number of false and missed alarms while keeping the impact of detecting intruders on the performance of the system to a minimum. The approach taken is through separation of the system in different layers, with elementary detectors at each layer. The observation of the detectors can then be aggregated to
determine occurrences of intrusions. Through separation of the system into layers, the authors concluded that their approach is more accurate than the elementary systems, has the desired ability to reduce the number of false and missed alarms and the effect of intrusion detection on the overall system performance. The proposed system [51] focuses on the traditional networking and intruder models in IDS; it does not deal with migrating software entities.

The agent paradigm has also found its use in Intrusion Detection Systems [52, 59]. Deeter et al. [59] aim at achieving bandwidth and performance, scalability, minimizing analysis delay as well as allowing integration with new IDS systems without the need for global change. As such, the mobile agent paradigm was adopted to establish a middle-ware layer allowing for different IDS to collaborate and secure a system. The mobile agents migrate to sources of potential attacks and determine whether to raise an alarm or not through analysis of data collected by various IDS. The mobility of agents adds virtualization and serialization overhead to the performance of the IDS architecture [59]. However, the system’s agents could potentially be used to carry out a denial of service attack on hosts, as the introduced IDS architecture was not designed to address agent security [59]. Section 3.2.1 details our contribution in securing mobile agent systems.

3.2.1 Distributed and Adaptive Security Monitoring through Agent Collaboration

The introduction of the proposed security scheme stems from the realization that agents interact in a distributed environment; hence, similarly, agent security needs to be validated in a distributed manner. As hosts monitor agents, data regarding the actions of the agent can be recorded. Our work is based on the assumption that there is a relationship, though not clearly defined, between the actions of an agent and the intent of such agent; whether the intent is malicious or not. The definition of the set of actions that can help determine whether an agent is malicious will vary from one host to another and such actions are herein referred to as threatening
actions. The consistent fact will remain however, that a malicious agent on one host is highly likely to represent a threat to the security of future hosts. Due to the variation in what constitutes a malicious agent, any proposed learning scheme must allow for such flexibility in identifying potential threats.

Our approach in tackling the problem is through the introduction of a variation of the Boosting-learning algorithm, namely Distributed and Adaptive Security-Monitoring through Agent Collaboration (DASAC). To determine whether an agent is malicious, DASAC relies on collaboration between the current host and past hosts visited by the agent. The current host acts as a decision maker; all hosts including the current one act as a base learner. We attain the required flexibility by allowing each host in the system, as base learners, to be trained independently and based on different feature sets. A discussion of what feature sets could possibly be used is deferred to Section 3.3. The base learners in DASAC are trained as follows:

• Implement a binary classifier, which can be a decision tree or any other classifier, where 1 is the class of a malicious agent and -1 otherwise.

• Train the classifier using a sample data set with the threatening actions of the host as the various features of each training instance.

Note that each host in the system may serve as a base learner and as a decision maker depending upon its contribution to the current decision-making process. The base learners, being trained independently, may implement various classifiers depending on the host’s administrator.

Upon arrival of an agent to a host, one of two cases may be true. The host may be seeing the agent for the first time or the host may have had prior experience with the agent. In either of these two cases, the host needs to determine whether to allow the agent to execute or not. If the host had no prior experience with the agent in question, it does not have any pertinent information about the agent to classify it as malicious or not, using its base learner. It must thus rely on the hosts that the agent has visited in the past, as specified by the agent’s migration history accessible
on an implementation-specific basis (see Section 3.3.1). If the agent had in the past executed on
the host, the host’s base learner can classify the agent.

Within DASAC, classification of an agent by the decision maker is based on the
following steps:

• If the host has had prior experience with the agent, the base learner of the host is used to
classify the agent; else, the agent is assigned to the default class of 0.

• The classification of the agent from every host in the agent’s history as determined by their
respective base learners are collected by the decision maker.

• Using the possibly diverse experiences of other hosts, the decision maker determines whether
to allow an agent to execute or not.

In essence, a Decision Maker (DM) interacts with the various base learners of the hosts in
the distributed environment to thwart attacks. In the final steps, a DM could use various
techniques such as majority-vote to reach a consensus. We however recommend a version of
weighted sum tailored to the problem at hand as specified in Equation 3.1 where \( \Psi_i \) represents the
class to which an agent has been assigned by the base learner of a host. We allow \( \Psi_i \) to possibly
have a value of 0 to ignore a base learner that does not have any information on the agent as such
may be the case for the learner on the current host. Furthermore, \( \tau_i \) and \( \lambda_i \), represent the trust, and
confidence levels, respectively, associated with each host being contacted.

\[
x = \begin{cases} 
1 & \forall \sum_{i=0}^{n} {\tau_i} * {\lambda_i} * {\Psi_i} \gt 0 \\
-1 & \forall \sum_{i=0}^{n} {\tau_i} * {\lambda_i} * {\Psi_i} \lt 0 \\
0 & Otherwise 
\end{cases}
\]

Equation 3-1
The recommended version of majority vote stems from our observations of the underlying mechanisms in inter-human collaboration. Consider the case where a person, $A$, asks a friend, $B$, for his/her opinion on a puzzling question; $A$ does not blindly believe $B$’s assertion. Instead, $A$ weighs his/her opinion and confidence on the topic with $B$’s recommendation based on two factors; namely, how much does $A$ trust $B$ and how confident is $B$ in his/her assertion.

The confidence level, in the proposed majority vote scheme, is determined by the accuracy of the classifier used in a host and varies between 0 and 100. The confidence of a host is communicated to the DM along with the classification of an agent.

The trust level, on the other hand, can be defined statically by the system administrator of a host based on the reputation of a particular host. We propose trust levels to be defined as values between 0 and 10. A default value can be specified for use whenever a remote host’s trust information is not available. Notice that setting the default value of trust to 0 would effectively allow the monitoring system to not take into account the experience/classification of unknown hosts. As the definition of trust levels does not carry over from one host to another, administrators are free in setting the limits of trust values in their systems.

If an agent is allowed to execute in the system, the decision maker keeps track of the actions of the agent. It can then periodically attempt to re-classify the agent and thus adapt to agents that may execute malicious code only on specific hosts. The frequency upon which to re-classify an agent is left as an implementation detail as it will vary upon the requirements of a host.

Although DASAC, as described, can be made to be completely autonomous, except during training, we understand that administrators may need to have hands-on control over whether or not an agent should be allowed to continue or start execution. To cope with such a need, we introduce the notion of Security Levels (SL) of agents on a host.
The SL of an agent is defined (Equation 3.2) as the ceiling of the product of its weighted-sum, as computed in Equation 3.1, and the number of security levels in the system ($\beta$). The result is divided by $\Delta$, representing the maximum sum of products multiplied by the number of cooperating hosts. Note that $\Delta$ is always greater than 0 as $n$ takes into account the current host as well. While the SL could be calculated for all possible value of the weighted-sum, one should note that it is not of importance when the weighted-sum is 0 or less as such agents have not been classified as malicious. Using the SL, the system can be made to be semi-autonomous, requiring human assistance once a threshold has been reached. Agent-human interaction can further increase the efficiency of the system as the agent can be made to adjust its classifier based on such interactions. Thus, DASAC may decide to use the collected data about an agent, classify it based on its interaction with an administrator, and inserts the information in the pool of training data. The classifier can be periodically retrained thereby leading to an adaptive security system.

\[
\Delta = n \ast \left[ \max(\tau_i) \ast \max(\lambda_i) \ast \max(\Psi_i) \right]
\]

\[
SL = \beta \ast \begin{bmatrix}
\sum_{i=0}^{n} \tau_i \ast \lambda_i \ast \Psi_i \\
\Delta
\end{bmatrix}
\]

Equation 3-2

Evaluation of our proposal requires its implementation in one of the numerous agent platforms that have been advanced in the literature (see Section 2.1.2). Addressing the security of the hosts further requires that an adequate security mechanism be in place to protect the agents as
well. The following section details our work in assessing the security in place in the agent platform chosen to evaluate DASAC, namely Aglet.

3.2.2 Secure Aglet Server

Prior to introducing any security schemes, we conducted some analysis of the security framework in place in Tahiti, the Aglet server, to ascertain that both entities, hosts and agents, can be properly secured. Our study has revealed the following vulnerabilities in the Aglet platform:

- Communication vulnerability established through the use of the Dsniff [60] package to intercept agents as they are being transmitted from one host to another. This has led us to conclude that the Aglet framework cannot satisfy the requirement of agents to migrate exclusively to intended hosts.

- Data vulnerability has not been addressed in the Aglet framework. While there is no acclaimed solution to the issue, it is imperative that agents be able to determine if their data has been tampered with, and determine the malicious host responsible.

- Resource vulnerability determined through analysis of the lifecycle of agents (see Section 3.1.1) in the environment. Through seemingly normal transition of states, agents can wreak havoc in hosts through repeated state transitions such as cloning.

Details on the aforementioned studies can be found in the Appendix. The results of our experiments served as the driving force behind the introduction of a new server namely Secure Aglet Server (SAS).

The Communication Layer of Aglets makes use of an unsecured protocol, thus leaving the framework open to a range of attacks. The current industry standard in addressing communication security being SSL, we opted to implement SSL in Tahiti, using the Java Secure Socket Extension (JSSE). Through the use of SSL sockets, we have endowed the Aglet
framework with the ability to communicate over secure channels capable of authenticating the parties involved, refusing unsecured connections and adjusting the security level on the channels. Through the availability of SSL in SAS, administrators will gain control of the level of security enforced on network links; most importantly, it provides a standard solution widely used in the industry to handle secure communication.

Granting Aglets the ability to detect tampering with their data required extended functionalities of the server. The Runtime Layer of SAS is extended to support the creation of Message Digests using the Java Cryptography Extension (JCE) as well as the ability to digitally sign objects. We provide Aglets with a java class library that implements the concept of Read-Only data. With the new functionalities of SAS in place, the library obtains a signed copy of the message digest computed by the host along with the host’s certificate. An Aglet can retrieve the message digests stored by the library and use the corresponding certificate to ensure that its data has not been tampered with. The introduction of computed message digests and digital signatures in the Runtime Layer of SAS provides Aglets with the capability of detecting active malicious hosts in the agent’s itinerary.

Dealing with the possibility of an Aglet overusing the resources of a host through seemingly normal transition between its lifecycle states, requires the design of a scheme to not only specify and track the resources in use by an Aglet but also to take proper actions once an Aglet attempts to overuse the host’s resources. The design of such a scheme led us to the introduction of MonitorAglet. Section 3.3.2.2 presents a study of the MonitorAglet’s response time when used as a DM in DASAC; while the study focuses on the classification of arriving agents, it showcases the small execution time overhead introduced by the MonitorAglet.

MonitorAglet is a static agent that monitors the state transitions of other Aglets in order to secure the system. The MonitorAglet agent tracks the number of instances of an Aglet, based on the Aglet’s properties such as the corresponding ID, to ensure that the specified limit is never
exceeded. Within the scope of SAS, we define instances as the instantiation of an Aglet or Message object. Definition 3.1: an Aglet $B$ is an instance of an Aglet $A$ if and only if one of the following is true

- $B$ belongs to the same resource object as $A$
- $A$ has created, retracted, or activated $B$
- $B$ is a clone of $A$.

Once an Aglet has reached its instance limit, the MonitorAglet prevents the creation, activation, or retraction of any other instances of the Aglets in the system until one of the instances has been deactivated, dispatched, or disposed of. As we attempt to protect hosts against malicious agents, we have introduced powerful capabilities to the Aglet framework. It is now possible to limit the number of instances of an Aglet executing on a host, thereby preventing the attack discussed earlier due to resource vulnerability of the Aglet platform.

In securing the Aglet server, we have introduced a new agent server, namely SAS, endowed with secured communication, ability to detect tampering of agent’s data along with prevention of over-usage of host’s resources. The Appendix provides further details on our approach in securing the Aglet server. We will now turn our attention to presenting the evaluation of our contribution in the following section.

### 3.3 System Evaluation

Having discussed our approach to addressing agent security, we present in this section the evaluation of DASAC. In doing so, we will start by presenting the implementation of DASAC in SAS (Section 3.3.1) followed by a discussion of DASAC’s evaluation based on parameters such as accuracy, missed alarms rates and response time (Section 3.3.2).
3.3.1 DASAC Implementation on SAS

We have previously addressed the motivation behind the presence of the MonitorAglet within SAS, which is to control the amount of resources being used by any particular agent (see Section 3.2.2). In implementing the DASAC scheme into SAS, the use of the MonitorAglet as the DM of a host was an obvious option, which we adopted. The base learner on the other hand is implemented as an independent agent for flexibility and performance. The implementation of the base learner as an agent allows for separate threads of execution to handle classification and training. Moreover, as agent entities, a host may have multiple base learners trained using independent feature sets; however within our implementation of DASAC we assumed that each host has one base learner. Furthermore, all base learners are built using the classifiers from the weka data-mining library [54].

Figure 3-2: Overview of DASAC Implementation in SAS

Figure 3.2 shows a pictorial representation of the interaction between DMs and base learners within SAS. The dotted lines in the figure denote the trajectory of the agent migrating from one host to another; the solid lines depict the communication between the DM and the base
learners in order to decide whether to allow the arriving agent to execute. In a nutshell, using the MonitorAglet as the DM of DASAC with other agents acting as base learners extended SAS. The implemented version of DASAC makes use of the suggested version of weighted sum (equation 3.1) to classify agents. Moreover we employed the notion of security levels, which are used to determine whether interaction with a system administrator is needed to dispose of an agent classified as malicious. Five distinct security levels were defined; the MonitorAglet is augmented with the capability of requesting interaction from a system administrator to determine the appropriate set of actions to undertake once an agent has been identified as a malicious entity below or at level 3. The malicious agent of levels 4 and 5 are automatically denied execution. The choice of level 3 is an arbitrary cutoff point that can be tuned by the system administrator of a host.

The set of features selected to train the base learners need to correlate in a manner that differentiates malicious entities from non-malicious ones in the system. Within SAS, agents are monitored and based upon the number of instances running, the MonitorAglet can allow or reject a requested action such as cloning, dispatching etc. Such actions of agents have been shown (see Section 3.2.2) to be potentially detrimental to the security of a system and thus constitute good candidates for inclusion in the feature set. It is our belief that the frequency at which an agent requests the right to transition from one state to another, along with the total amount of time spent on the host, can help in identifying malicious actions. Due to Java’s sandboxing techniques, a security manager prevents access to entities lacking the proper access rights to local resources. The malicious agents may attempt to access various resources in the hope that the security policy in effect is not well defined. As a result, we take into account the frequency at which an agent generates security exceptions. The security manager resides within the Runtime layer of SAS and in order to track the security exceptions generated by an aglet, we introduced a listener class through which interested parties can be notified whenever such events occur.
To sum up, we selected the features in table 3.1 to train the classifiers based on CPU usage times, lifecycle state transitions, and access to low-level system resources. Once the feature set has been selected, we trained the classifiers to extract data patterns that may help classify an agent based on its behavior. We used a system consisting of 8 hosts. For simplicity, hosts 1 thru 4 have different classifiers, namely, an alternating decision tree with 0 boosting iterations, a fast decision tree learner, a decision stump, and a naïve Bayes classifier. The remaining hosts (5, 6, 7 and 8) employ the alternating decision tree as well but with 3 boosting iterations; their results is averaged and represented during our experiments as Host 5. The implementations of the classifiers used are from the weka library [61]. Moreover, the first four hosts have different trust levels, with all others having the same trust levels.

Table 3-1: Classification Features for Each Agent.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Feature Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Biased Running Time</td>
<td>Biased Length of Execution Time on System</td>
</tr>
<tr>
<td>Cloning Frequency</td>
<td>Ratio of number of cloning to average time between cloning</td>
</tr>
<tr>
<td>Activation Frequency</td>
<td>Ratio of number of activation to average time between activations</td>
</tr>
<tr>
<td>Dispatch Frequency</td>
<td>Ratio of number of dispatching to average time between dispatching</td>
</tr>
<tr>
<td>Retract Frequency</td>
<td>Ratio of number of retraction to average time between retractions</td>
</tr>
<tr>
<td>Arrival Frequency</td>
<td>Ratio of number of arrivals to average time between arrivals</td>
</tr>
<tr>
<td>Security Access Request</td>
<td>Ratio of number of security access requests to average time between such requests</td>
</tr>
<tr>
<td>Security Access Grant</td>
<td>Ratio of number of security access grants to average time between such grants</td>
</tr>
<tr>
<td>Security Access Denial</td>
<td>Ratio of number of security access denials to average time between such denials</td>
</tr>
</tbody>
</table>
Due to the fact that training data for classifiers are not readily available, we trained our classifiers using a data set generated by tracking the features of interest during actual runs of the following agent-based applications:

- MAMDAS [62]
- Private Information Retrieval prototype (see the Appendix)
- Instance and Message DoS attack generator agents (see the Appendix).

The choice of applications used to collect the data was based not only on their availability, but also due to the fact that their classification is known a priori, as discussed in SAS (see the Appendix). Furthermore, the choice of applications encompasses both classes of agents, malicious and benign, that are of interest to our work. The MonitorAglet tracks every event generated by agents in the system and construct a data sample for the corresponding agent. The constructed sample consists of the features that we identified as having the potential to help identify malicious entities (see Table 3.1).

The generated data set consists of over 3000 samples, manually classified, of which, roughly 15% is randomly chosen to train each classifier. The classifiers used the remainder of the data set for accuracy assessment. The accuracy of each host classifier is used as the confidence level of the host in question, as suggested by DASAC, in all our experiments.

Figure 3.3 presents the accuracy rate of the hosts in the system over multiple runs. The accuracy of hosts shown was determined based on the ability of the host’s classifier to properly classify the remainder of the 3000 samples in the data set not used for training. As we noted earlier, the hosts use different classifiers resulting in different accuracy rates based on the strength of the classifier being used. As thus, Host1 exhibits the worst accuracy as it uses an alternating decision tree with 0 boosting iterations. Through the use of decision stumps, naïve Bayes classifier and alternating decision trees with numerous boosting iterations, the other hosts are able to achieve a higher accuracy rate.
Experimental Results

DASAC is evaluated based on several performance metrics such as accuracy (i.e., the ability of the framework to properly classify agents), the false and missed alarm rates and the time it takes to evaluate whether an agent is malicious. The results of such evaluations are herein presented starting with the system’s accuracy and trust impact in 3.3.2.1.

3.3.2.1 System Accuracy and Trust Impact

In evaluating DASAC, we first focused on measuring the accuracy of the system in identifying malicious agents and the impact of trust level on the system’s accuracy. Our primary goal is to study DASAC’s effect on the ability of a host to identify malicious agents. We chose the first host in the system (Host1 from Figure 3.3) as the local classifier against which DASAC
will be compared. This choice was based on the fact that Host1 showed a slightly better performance than random guessing. As such, our goal is to analyze how DASAC can help improve the performance of a host, through collaboration. We thus proceeded to launching the agent applications used in training the system classifiers. We also introduced a PortScannerAglet that repeatedly attempts to connect to numerous ports in the system whether it has access to conduct such actions or not. Furthermore, we manually created, dispatched, and retracted each of the following agents: CirculateAglet, WebServerAglet and HelloAglet. These agents are available as part of the Aglet framework. Lastly, we created and used a MigratingWebServer agent that migrates to hosts and attempts to set up a server on random ports, restricted or not, repeatedly. The MigratingWebServer agent migrates to a new host, once it has been denied access to ports over 10 times. Our reasoning behind the introduction of new Aglets that were not used during the training phase is to gain insights into the ability of our classifiers to perform well even in the presence of previously unseen behaviors.

The DM of each host dynamically classifies agents to determine whether or not they are malicious. We evaluated the system based on the accuracy with which the local classifier and DASAC recognize malicious agents. We also tracked the best accuracy recorded in the system and computed the average accuracy of the hosts including Host1. While the confidence levels used in the experiments were as described in Section 3.3.1, the trust levels, on the other hand, were assigned randomly. The local classifier is assigned a trust level of 9, close to the maximum of 10, to reflect the trust that we expect administrators to have in their own systems.

Figure 3.4 depicts the measured accuracy of DASAC compared to that of Host1’s classifier, the best accuracy rate measured in the system, and the average accuracy rate of hosts in the experiment. On average, one can conclude that DASAC’s performance lies between the average accuracy of the involved hosts and that of the most accurate classifier. The fluctuations in the accuracy rate measured are due to the fact that the hosts are independently trained for every
experimental run on a random sample. Thus, their performance, is slightly dependent on the sample used during training.

While DASAC generally outperforms the worst classifier in the system, it matched the first host’s performance during the first run of the experiment. The only reasonable explanation for such a poor performance by DASAC stems from the trust levels used during the first experiment. We noted that the total trust levels of hosts 2 thru 5 varied from one experiment to the next as follows: 5, 16, 23, 27, and 36. When the trust levels of the other hosts are low compared to that of the local host, DASAC’s performance seems to be more dependent on that of the local classifier. This brings us to the second set of experiments that were carried out to further investigate the effect of trust levels on DASAC. During the second experiment, we kept the trust levels of all hosts, including Host1, identical. The trust levels were however varied from one experimental run to the next starting at 0 up to 10.

Figure 3-4: Local Vs. DASAC Accuracy.
Figure 3.5 presents the results of the second experiment. The figure shows that DASAC still outperforms the average accuracy rate computed. The experiment revealed two crucial points; Firstly, all the classifiers in the system can indeed outperform DASAC, as is the case when the trust levels are 0. The explanation behind such an occurrence is due to the fact that DASAC will classify all samples as 0, which is in effect non-malicious. Thus, DASAC will fail to classify any malicious agents possibly degrading to an accuracy rate of 0. The second interesting point is the fact that DASAC’s accuracy seems to quickly become dependent upon the accuracy rates of the best classifiers in the system. This is justified by the fact that DASAC is in effect designed to take advantage of the strength and experience of other hosts in the system. Once the trust levels are identical for all hosts, the only determining factor in classifying an agent becomes the confidence of hosts. The more confident hosts have a more significant weight on the system’s classification of an entity.

![Figure 3-5: Effect of trust levels on DASAC Accuracy.](image-url)
The experiment allowed us to assert the accuracy of the herein introduced framework and analyze the impact of the use of trust levels on the observed accuracy. The next step that we undertook was to analyze the response time of DASAC in determining whether to allow an agent to execute along with the false and missed alarm rates; the results of the experiment are presented in Section 3.3.2.2

### 3.3.2.2 Response Time, False and Missed Alarms Rate of DASAC

As noted earlier, determining the false and missed alarm rate of an IDS system is crucial in evaluating the system (see Section 3.1.2). Furthermore, the time that it takes to determine the classification of an entity needs to be as low as possible, since a slow response time would allow attackers to quickly execute their malicious code and migrate to the next target. Keeping in line with such facts, the focus of this set of experiment was geared towards evaluating the accuracy, missed and false alarm rate of DASAC along with the response time of the framework. In setting up the experiment, we decided to use two hosts and collect the information of interest to our study. The two hosts used in the experiment are Host1 and Host2 from Figure 3.3 with training accuracy of 55.4% and 86.2%, respectively. Furthermore, we designed a new Aglet whose sole task is to execute on Host2 and repeatedly create agents and dispatch them to Host1. The intent behind such a setup is to study the response time of the framework under ideal conditions when the agents are arriving and having been to one and only one host prior to the current one. Restricting the study to the ideal condition will mostly affect the response time of the system while allowing us to identify any possible performance bottlenecks. Moreover, the experiment is intended to allow us to study the efficiency of the system when it is collaborating under the aforementioned conditions.
As we have mentioned, agents are created on Host2 and dispatched to Host1 (one-hop away). The agents that are being dispatched are the DoS attack generators noted earlier (see Section 3.3.1), the PortScannerAglet, and the MigratingWebServer along with the followings from the Aglet platform:

- examples.simple.DisplayAglet
- examples.hello.HelloAglet
- examples.itinerary.CirculateAglet
- examples.mdispatcher.HelloAglet
- examples.http.WebServerAglet
- examples.talk.TalkMaster

The choice of such agents was motivated by the fact that they constitute a fair representation of the types of agents that can exist in the system in terms of their intentions being malicious or not. The agent executing on Host2 continuously generates and dispatch the agents to Host1; from that location, we measure the accuracy, false and missed alarm rate, and the time it takes for Host1 to contact Host2 and determine whether to allow an arriving agent to execute. The key point here is that as DASAC is currently implemented, Host1 needs to formally deploy an Aglet to Host2 and collect the information necessary to classify any arriving agent.

The result of the experiment is presented in Figure 3.6 showcasing the accuracy of DASAC on Host1, along with the false (FAR) and missed (MAR) alarm rates of the host. As expected, DASAC outperforms Host1 and draws closer to the accuracy of Host2 while the false and missed alarm rates decrease before settling.
The average amount of time required to classify an arriving agent was also measured. As expected, the ratio of time spent on communicating with Host2 compared to the total time required to classify arriving agents was drastic as shown in Figure 3.7. The figure shows that the

Figure 3-6: False Alarm Rate, Miss Alarm Rate and Accuracy Rate.

Figure 3-7: Processing Vs. Communication Time of Arriving Agents.
communication time represents over 90% of the time it takes to determine whether to allow an agent to execute. Such an observation is a direct result of the fact that the current implementation of DASAC has to dispatch an Aglet to past hosts and collect the required information to classify an agent. We expect the communication time to be linearly dependent upon the number of past hosts that must be visited. In order to avoid such a significant cost in classifying an agent, we slightly modified our implementation of the framework in DASAC.

The system was modified by attaching the address of the current host to agents being dispatched along with pertinent information that the framework requires such as the class to which the agent has been assigned prior to migrating from the current system, along with the confidence factor of originating host. Encoding such information is done in the Runtime layer of the platform to avoid alteration or tampering with such data by malicious agents. Having altered our implementation of DASAC on Aglet we noted close to a 90% reduction in the response time of the Host1, as one would expect. Such a drastic reduction is due to the fact that Host1 no longer has to use network bandwidth to collaborate with Host2, nor does it have to wait for such information to be available before making a decision. With the changes in the framework’s implementation, the average response time of the DM in classifying agents is effectively reduced.

3.4 Conclusion

We have introduced a novel distributed and adaptive security-monitoring framework achieved through agent collaboration across multiple hosts. Furthermore, the security of the Aglet server was analyzed, and led to the introduction of SAS which provides secured communication; ability for agents to detect tampering of their data, and ability for hosts to restrict the actions of malicious agents that may lead to denial of service attacks. DASAC has only been implemented within SAS at this point, it can however be easily applied to any agent platform. The framework,
as we have shown, builds on the idea of boosting to allow host protection by classifying agents based on their reputation. The system is flexible enough to support the incorporation of various classifiers that may be trained using independent variables, as the hosts do not communicate their feature sets to each other. Moreover, DASAC introduces the notion of security levels to support human-agent interaction in order to render the system even more flexible and robust. As a whole, our work has addressed agent security by focusing on the threats as opposed to their origins; this has led to our proposal in securing both agent-based system entities, agents and hosts.

DASAC’s weakness lies in its dependence upon the choice of classifiers and feature sets used to train the classifiers. Future work on the subject should analyze agent patterns in more detail especially as more agent applications become available. One possible approach to addressing the issue may be to reduce the scope of the problem and study agent patterns based on specific class of applications. As such, administrators will be better equipped in choosing a feature set along with classifiers that may be used on a host based on the services such a host may provide.
Chapter 4

Reconfigurable and Interoperable Sensor Network

Traditional Sensor Networks consists of multiple resource-constrained devices or nodes, deployed in an area of interest in order to collect and relay information to a base station for processing. Once deployed, the nodes generally react to pre-defined environmental stimuli, collecting data of interest. The environment in which sensor networks are deployed is inherently dynamic, yet the nodes are statically tasked. As node programming generally occurs pre-deployment, neighboring or overlapping networks are unable to interoperate with each other either to share data and/or resources for the purpose of dealing with common issues such as routing efficiency or power consumption. Pre-deployment programming leads to isolated networks, consequently, every new application would require deployment of a new network thereby contributing to node proliferation in the environment. Despite the typical constraints within which sensor networks need to operate, sensor network applications may have real-time processing requirements. Due to the possible timing constraints on the network’s data processing ability, the nodes reaction to environmental stimuli should be swift. Furthermore, processing of the collected data should occur in a timely manner.

The use of sensor networks in various applications is increasing [48], thusly exposing the need for dynamic tasking of sensor nodes, interoperability of networks, and timely reaction to environmental stimuli. In addressing the aforementioned needs of sensor network, our approach relies on the use of Field Programmable Gate Arrays (FPGAs), along with mobile agents to introduce a Reconfigurable and Interoperable Sensor Network (RISN) framework. RISN aims at accomplishing four main goals, which are:
• Increased processing power through the use of hardware accelerators to help support time-constraint applications

• Ability for the framework to interoperate with existing networks in the area and thus enabling data-sharing while preventing node proliferation

• Dynamic tasking of nodes to support changing needs of users and applications through the use of mobile agents, and

• Service Provision to efficiently support common needs of applications

In further detailing the RISN framework, proposals relevant to this discussion will be introduced in the next section (4.1). Section 4.2 will discuss the design and prototype implementation of RISN, while the framework’s evaluation will be presented in sections 4.3. Section 4.4 will conclude this chapter highlighting our contribution.

4.1 Background

The framework herein introduced draws from various research fields in accomplishing its goals. Issues pertaining to mobile agents, sensor network, and target tracking in sensor network have been discussed in sections 2.1, 2.2, and 2.3, respectively. Furthermore, dynamic tasking in sensor network was introduced in section 2.4. The focus in this section is thus restricted to a brief discussion of proposals related to our research in order to ensure a deep understanding of the underlying techniques and motivations that led to the introduction of RISN.

As we rely on vision-based target tracking to evaluate various portions of the RISN prototype, we feel compelled to herein present the state of the art in that area of research. Vision-based tracking aims at generating an object’s trajectory through analysis of the objects location in successive frames of a video. Within sensor network, proposals to deal with tracking have generally focused on tracking objects while attempting to minimize power consumption in the
network (see Section 2.3). It has been noted that applications in Sensor Network in general need to react to stimuli in a timely fashion to avoid relaying or processing obsolete data [63]. Following that trend, VigilNet [64] was introduced to possibly track fast moving target, such as a vehicle, within specified timing constraints by dividing the real-time constraints into sub-constraints that are attached to particular partitions of the task at hand. The term Video Sensor Network was used in [65], to denote a set of nodes equipped with cameras, which capture and relay frames of videos to a base station. The video sensor network relies on real-time processing of the data and thus generally uses in-network processing to reduce communication volume and latency. Through the use of dynamic critical path task mapping and scheduling, video sensor network attempts to map and schedule tasks while minimizing energy consumption within the specified delay constraints boundaries.

Yilmaz et. al categorized the methods used to visually track objects as Point, Silhouette, or Kernel based [66] depending on the representation of the object by the tracker. The Mean-Shift tracking algorithm is a kernel-based tracker introduced by Comaniciu et al [67, 68]. As a tracker, Mean-Shift algorithm locates objects by iteratively maximizing the similarity of color distributions between a target and a candidate location in successive image frames. Due to the algorithm’s ability to handle partial occlusion, work in the presence of clutter and its ability to adapt to changes in the size of the object being tracked, it is relied upon to evaluate the proposed platform (see Section 4.4).

4.2 RISN System

The aim of RISN is to promote sensor network interoperability and allowing efficient dynamic tasking of nodes. RISN utilizes a hardware overlay atop surrounding traditional sub-networks, with the overlay capable of communicating with underlying sub-networks. To achieve
such goals, RISN is, in essence, a deployed network made up of FPGA nodes with increased processing abilities than that of traditional sensor networks. FPGA allows for a hardware chip to be reprogrammed in order to support a particular application. As the RISN nodes are FPGA-based, it is expected that they will be more expensive than traditional sensor nodes, thereby limiting its application in traditional Sensor Network environments. Consequently, this dissertation strays away from the traditional assumption that nodes are very cheap and thousands are easily affordable by an individual or company for a specific use. However, a hardware overlay network with computationally powerful nodes, with greater communication range, would only require the deployment of a few RISN nodes. Being that the overlay network will consist of a small set of nodes, collection of such nodes in the event of failure or to replace their temporary power source is feasible especially in urban settings. As FPGAs are reconfigurable, the overlay network as a whole can be reconfigured to adapt to changes in the underlying networks such as the introduction of a new set of nodes or a new communication protocol. The re-programmability of nodes and the speedup in execution are attractive features of FPGA-based nodes, which RISN aim to exploit.

To abstract the underlying heterogeneity of sub-networks from user applications, the work herein discussed requires a uniform representation of data in the system, and a common communication medium amongst nodes in the overlay, which may differ from those of surrounding sub-networks. Consequently, application developers can focus on the task at hand instead of dealing with potential issues resulting from formatting or communication heterogeneity. RISN extends the traditional definition of sensor network by viewing the network as a system with four interacting entities namely (i) the RISN hardware-overlay, (ii) the sub-networks, (iii) a Base Station, and (iv) the user agents. The interaction between the four entities is aimed directly at improving data availability to the user and allowing the latter to process harnessed data efficiently. Figure 4-1 depicts RISN acting as an overlay to surrounding networks
thereby presenting a unified interface to data available from RISN and the sub-networks to interested parties.

4.2.1 RISN Overlay

The RISN Overlay is at the core of the RISN system itself. It is comprised of FPGA nodes encompassing a General-Purpose Processor (GPP). The GPP supports a generic set of instructions to allow software re-configurability, while the FPGA accomplishes the same feat at the hardware layer. As the nodes in the overlay are FPGA-based, the ML405 evaluation board
available from Xilinx are utilized to develop a RISN prototype. The board contains a
PowerPC405 microprocessor that is used as the GPP. The Serial and Ethernet connections of the
board were activated in the implementation. The Ethernet connectivity is used primarily to
support communication between overlay nodes. Managing the different hardware components in
the system is a small footprint of the Linux kernel (1,767KB) running on the board. Figure 4-2
attempts to capture a snapshot of the various components of RISN node and their primary
interactions. The overlay nodes encompass 5 major hardware/software components, which are:

- Agent System
- Service Architecture
- Low-Level Tasks (LLT)
- Interoperability Interfacing System (IIS)
- Local Sensors

Figure 4-2: Components of RISN Overlay Node
4.2.1.1 Agent System

Agents are prime candidates to allow deployment of user applications to remote systems and are thus adopted by the RISN system as a whole. The ability for agents to migrate to a node and perform user tasks is in line with RISN’s requirement of supporting dynamic tasking of nodes. The agents do not migrate in the sub-networks but instead they migrate in the overlay network, the Base Station and the user devices. Note that there is an underlying assumption that agents can migrate to the overlay node of interest and execute their code.

The Agent System is intended to provide the interface between overlay nodes and user applications. The choice of an agent platform to use was guided by the rankings of such platforms as conducted by Altmann et. al [10]. The dissertation thus relies on the use of Aglets [16], as the platform of choice. The execution of the Aglet server on the ML405 board was made possible through the use of the Sun Headless Java Runtime Environment for the PowerPC processors running Linux distributions. Upon migrating to a node, the Aglets can use the resources available on the overlay node, along with the GPP to perform the task at hand. Every node in the overlay network contains a Static Service Agent (SSA) to allow migrating agents to discover available resources on the hosting node. Further details on the SSA are provided in the following subsection.

4.2.1.2 Service Architecture

The data available at the overlay nodes can be used to perform various functions, depending on the sensing abilities of the overlay node and its underlying sub-networks. A subset of such functions can be provided as services to user-applications, especially if there exist optimal implementations of the functions that may benefit the system. RISN requires the presence of a
service-based architecture in the overlay nodes to deal with that possibility. Implementation of the service architecture must enable users to discover the exact services available in the system and on any particular node. While the Base Station addresses the issue of detecting the services available in the system (see section 4.2.2), the latter requirement is met through provision of such information to local user applications using the SSA. The interaction between the SSA and users is accomplished through exchange of the following messages:

- **GetAvailableServices**: returns a list of the service names that are available to users on the current node. It is assumed that users know in advance how to interact with the services based on their names.
- **GetServiceHandler**: returns a ServiceHandler object associated with a particular service name. The object returned can be used to access the functionalities offered by the service.

As the SSA supports these two messages, the services in the system can be discovered and make use of one another through the SSA; thereby allowing complicated services to be built from simpler ones. The ability for services to make use of each other’s functionalities is one of the requirements of service-based architectures, herein handled through messages.

The ServiceHandler object, is an abstract class through which users are guaranteed the ability to access the following functions:

- **GetServiceName**: retrieves the name of the service provided by the handler
- **GetServiceMetaData**: retrieves the metadata of the service being provided. Currently, the metadata is simply a string that must be parsed by user agents.
- **Exec**: Instructs the handler to execute a particular command on an input object

The services that an overlay node can provide is limited solely by the node’s sensing ability, and that of its underlying sub-networks. Every overlay node provides communication and, when applicable, data collection services. Communication services allow users to send data to as well as receive data from underlying sub-networks. Data collection services provide access to
data from local sensors if existent. Within the prototype, the following functions of the ServiceHandler have a default implementation that can be overridden by IIS and local sensor handlers in order to provide Communication and Data Services:

- **GetRawData**: retrieves data from an underlying channel (local sensor or IIS).
- **SendData**: allows data to be sent to subnets or, if necessary local sensors.

Other services that allow user applications to take advantage of the processing ability of LLTs are highly recommended though not required in implementations of the system. One should however note that the interaction of user applications with nodes’ LLT is expected to occur solely through services; consequently if the abilities of LLTs are not presented as services, they may be a waste of FPGA resources. The SSA is primarily intended to provide an interface for users to access LLTs, IIS and local sensors to support RISN aim at yielding an interoperable sensor network with efficient processing of data.

### 4.2.1.3 Low-Level Tasks

Under the assumption that the set of data streams that any sensor node deals with limits the possible applications that such a node can support, the notion of Low-Level Tasks (LLT) is introduced. If the data streams of a node are all temperature readings, the node in question will primarily be monitoring changes in temperatures, and converting from one metric to another. On the other hand, if such data stream originates from cameras, the node may be involved in tracking or object detection. In short, the “sensing” ability of a node limits the type of operation that the node will most likely be involved in. Thus, this dissertation contends that common tasks of applications can be abstracted and incorporated into the FPGA hardware to improve efficiency. Common tasks of the node are referred to as Low-Level Tasks (LLTs) due to the fact that they are defined at a fine granularity and are dependent upon the set of applications that a node can be
reasonably expected to deal with. LLTs can implement common and frequently used tasks in hardware, thereby allowing users the opportunity to efficiently process data of interest.

Similar to traditional sensors, data collection is a primary task of RISN, in addition, providing near optimal implementation of functions common to applications is also an integral part of the system. In light of this observation, the LLT system in our prototype is built with array processing in mind providing users with the ability to perform various arithmetic operations on large or singular arrays. The LLT arithmetic operations are based on IEEE-754 single precision specification, with the ability to convert to and from 32-bit integers. The LLT also provides users with the ability to apply the Normal Kernel to an array of values as well as compute the histogram of an image based on a specified number of bins. The logic behind provision of such operations in the library lies in the fact that this dissertation targets image-processing applications on the RISN nodes, as thus such operations were extracted as the most basic commonalities. The LLT driver works with images of size 640X480 in computing histograms; and supports up to 64 histogram bins per color channel. Furthermore, the driver supports the remaining operations of the LLT with each operands containing up to 229,376 elements. The driver relies on a 3MB memory pad to perform its duties. While the aforementioned capabilities of the LLT are hardware-based, they can be combined to provide support for higher-level functionalities. The challenges faced, along with the approach taken to a floating point LLT (section 4.2.1.3.1) and the Normal Kernel (section 4.2.1.3.2); lastly, the potential benefits of implementing a task in hardware were studied and presented in section 4.2.1.3.3.

4.2.1.3.1 Floating Point LLT

Floating Point operations in hardware can be quite expensive; in terms of the size and space requirements. However, the prototyped LLT needs to frequently perform floating-point
operations, thus the need to supply a Floating Point Unit (FPU) capable of handling such operations. There is a trade-off between size and performance in designing such an FPU. As per the ultimate goal of using the FPU in a low-resource environment, priority has been given to minimizing the size of the FPU. The FPU supports the following operations in IEEE-754 single precision format:

- Addition/Subtraction
- Multiplication
- Comparison
- Division
- Square Root
- Conversion to IEEE-754 single precision
- Conversion from IEEE-754 single precision

Furthermore, as noted earlier, the generation of the histogram of an image is an integral component of the FPU. Computing the bin to which a pixel belongs to is at the core of generating histograms for the corresponding image. In general, computing the histogram address of a pixel is highly dependent upon the number of bins the histograms in the system are allowed to have. The FPU supports histogram generation of 2, 4, 8, 16, 32, 64, 128 and 256 bins per color channel with the assumption that we are operating in the RGB-24 color space. Note that, for one component of unsigned 8-bit value, if the histogram is made of 256 bins, the binary representation can be used as the index into the histogram bins. Similarly for 128 bins, the leftmost 7 bits of each color component represent the address of the component into the histogram and so on for any number of bins that is a power of 2.

\[
A = (R \gg n) \times 2^2 + (G \gg n) \times 2 + (B \gg n)
\]

Equation 4-1
The histogram address, $A$, of a pixel is thus obtained using Equation 4-1 where “$\gg n$” representing the right-shifting operation of the 8-bit value of the component. Let $p$ be the number of bits required to represent the number of bins, $K$, in a component; $n = 8 - p + 1$. Thusly, for 256 bins, $n = 8 - 9 + 1 = 0$; hence no shifting is required; whereas for 2 bins, $n = 8 - 2 + 1 = 7$. By using this scheme, one simplifies the process of computing the mapping from a pixel to its histogram bin. Since $K$ is a power of 2 and is known in advance, the properties of binary multiplication and addition can be relied upon to compute $A$; essentially free of any hardware overhead, a considerable achievement in helping keep the size of the FPU to a minimum.

4.2.1.3.2 Hardware Implementation of the Normal Kernel

The normal kernel is dependent upon the exponential function; which makes it very undesirable in terms of computational time. The Normal Kernel implementation relies on the work of Schraudolph [69], which approximates the exponential function $e^y$ for $y$ being an IEEE-754 double precision floating-point number. As 8 bytes are used to store a number in double precision; computing $e^y$ is achieved by setting the leftmost 4 bytes to $i = ay + (b - c)$ where $a = 220 / \ln(2)$, and $b = 1023 \times 220$ and $c = 60801$ is an adjustable parameter affecting the accuracy. The value of $c$ stated here has been shown to minimize the root-mean-square relative error [69]. Furthermore, the algorithm is only applicable if the value of $y$ is roughly between -700 and 700.
Implementation of Schraudolph’s work is not directly applicable to the goal of providing support for the normal kernel in hardware. However, the value $b - c$ can be pre-calculated, thus only two operations are required (one addition and one multiplication). Furthermore, the addition and multiplication are conducted in single precision. 896 is added to the exponent portion of the adder’s result (bits 30 to 23) yielding the 11-bit exponent in IEEE-754 double precision. The sign of the result is maintained along with bits 22 down to 6. This scheme renders Schraudolph’s work applicable to the task of approximating the exponential function in hardware using single precision. Equation 4-2 derives the application of Schraudolph’s work to the Normal Kernel. As shown, the implementation of the Normal Kernel to any value $x$ in hardware is thus reduced to one multiplication, one addition and one division. This is feasible since the coefficient of $x$ and the second addend can be pre-computed for different dimensional spaces $d$ and passed down to the hardware. Applying $K_N(x)$ essentially comes free of charge as the LLT already encompasses an adder, a multiplier and a divider in IEEE-754 format. The exponential function approximation is only valid for numbers in the range from -700 to 700. This limitation is suitable for the purpose in computing the color probabilities, since the value $(y-x_i)$ in the Mean-Shift algorithm is less than the kernel bandwidth $h$ [67, 68]. Thus, application of the kernel only operates on values within the specified range.
4.2.1.3.3 Study of LLT Efficiency

In order to study potential benefits of performing a task in hardware, the Mean-Shift algorithm is implemented as a LLT. The Mean-Shift algorithm has been implemented as two major components; notably a Mean-Shift Controller and a Memory Controller. The Memory Controller is needed to provide interaction to the data available in the system. It thus provides read and write functionalities to the system’s memory through a Processor Local Bus (PLB) connection, thereby allowing access to data. The remainder of the system relies on the FPU and normal kernel implementations described earlier (see sections 4.2.1.3.1 and 4.2.1.3.2) to achieve its task. The FPU tightly coordinates with the Mean-Shift Controller, in charge of computing memory addresses to be accessed along with coordinating the computations to be performed to locate the target of interest. A general depiction of the implementation is given in Figure 4-3.

Upon initiation of the system, the Mean-Shift controller computes the Normalization value (NG) used to normalize the color probability distribution (HRG). The new location of the target is then determined in the Location Generator (LG), which is used to update the color probability distribution (NG and HRG). The updated distribution determines whether or not the computed location should be adjusted. If no adjustment of the new target location is needed, the system terminates. The Mean-Shift algorithm [67, 68] specifies the terminating or adjustment conditions of the system.
4.2.1.3.3.1 Mean-Shift Controller

The Mean-Shift Controller, or General System Controller (GSC) represents the core of the hardware implementation. GSC performs various functions such as kernel evaluation, generation of the normalization constant, histogram and Rho values as well as determining the candidate locations of the target. Moreover, the controller determines the addresses in memory to be accessed and deciding when to initiate the hardware or terminate the search for a particular target. The controller manages various components including the Floating Point Unit.

The Normalization Generator (NG) System, as the name suggests determines the normalization value to be used to generate the color probability distributions. In general, the
normalization value is defined in the Mean-Shift algorithm as per Equation 4-3, where \( y \) is the pixel representing the kernel center of the target candidate, and \( k \) represents the kernel profile.

\[
C_h = \frac{1}{\sum_{i=1}^{n_h} k\left(\frac{y - x_i}{h}\right)^2}
\]

Equation 4-3

Furthermore, \( x_i \) represents the current pixel being processed and \( h \) is the bandwidth. The normalization value is intended to calculate the probability color distribution given by Equation 4-4 with \( b(x_i) \) mapping the pixel at location \( x \) to the corresponding histogram bin \( u \), and \( \delta \) representing the Kronecker function [70]. Taking the last point into account, the NG System takes advantage of the need to calculate the summation for both the normalization value and the color distribution. The numerator and the denominator of the probability color distribution in the NG system were computed, while delaying the actual division to yield the distribution to another system, notably HRG. Delaying the division is justified by the fact that the value of \( C_h \) is needed prior to computing the normalized distribution. The Normalization system works by accessing memory and reading the pixel value \( x_i \), simultaneously, the row coordinates of pixels \( y \) and \( x \) are converted to floating point format. The converted coordinates of \( y \) and \( x \) are used to compute the squared Euclidean norm specified in Eq. 4-4.

\[
\hat{p}_u(y) = C_h \sum_{i=1}^{n_h} k\left(\frac{y - x_i}{h}\right)^2 \delta[b(x_i) - u]
\]

Equation 4-4

The result of these steps yields the value on which the kernel is applied. The summation of the kernel results yields the normalization value. Addition of the kernel results to the
appropriate histogram bins, as determined by the pixel read, yields the numerator of the
probability color distribution.

\[
\rho[p(y_0), q] = \sum_{u=1}^{m} \sqrt[p]{p_u(y_0) q_u}
\]

with \( p_u(y) = C_k \sum_{i=1}^{n_h} k(\frac{y - x_i}{h}) \delta[b(x_i) - u] \)

and \( q_u = C \sum_{i=1}^{n_h} k(\|x_i\|_2^2) \delta[b(x_i) - u] \)

Equation 4-5

The Histogram and Rho Generator (HRG) System has the dual task of normalizing the
probability color distribution and computing the Rho value presented in Equation 4-5. The system
works by reading pixel values of the target model, determining their histogram addresses, and
reading the histogram values computed by the NG System. The histogram values read are then
divided by the Normalization factor generated by NG before being written back to that same
histogram bin. The result of the division is also multiplied by the histogram value from \( \hat{q}_u \). The
Rho value is computed through summation of the square root of the multiplier’s result. Once all
values have been processed, the accumulated value is obtained and GSC can then proceed
towards activating the Location Generator System.

The Location Generator (LG) System computes the weights associated with each pixel
along with the new location of the target as specified in the Mean-Shift algorithm. This approach
was adopted for memory efficiency as it eradicates the need of saving the weight associated with
each pixel. Instead, the weight of a pixel is computed on the fly, while the kernel is being applied
to the pixel’s location, as in the NG System. As pixel values are read, the histogram address of
the pixel is generated, and used to access \( \hat{q}_u \) and \( \hat{p}(y_0) \) in order to perform the steps outlined in Equation 4-6 which computes the weight of the pixel.

\[
w_i = \sum_{u=1}^{m} \delta[b(x_i) - u] \sqrt{\frac{q_u}{p_u(y_0)}}
\]

Equation 4-6

Once the kernel has been obtained, the computed weight is multiplied by the result of the kernel. The result of the multiplication is used to maintain the summation forming the denominator of Equation 4-7. The pixel’s location coordinates also use the result of the multiplication to generate the numerator through further multiplication. Once all the pixels have been processed the summation of the rows and columns are divided by the denominator that has been computed and saved yielding the row and column values of \( \hat{y}_1 \). The implementation of GSC and of the Mean-Shift algorithm in general, as presented, focuses on speed rather than hardware size as it is intended to study the potential speedup in execution time that LLTs can provide to applications.

The interaction of the two subsystems, namely GSC and the Memory Controller, provides the tracker implementation used to evaluate the potential benefits of LLTs. The following section presents an evaluation of the proposed system highlighting its swift response time to stimuli.

\[
\hat{y}_1 = \frac{\sum_{i=1}^{n_k} x_i w_i g(\frac{y_0 - x_i}{h} \bigg| \bigg| y_0 - x_i \bigg| h)^2}{\sum_{i=1}^{n_k} w_i g(\bigg| y_0 - x_i \bigg| h)^2}
\]

Equation 4-7
4.3.1.3.3.2 LLT Efficiency Evaluation

Evaluation of the system to determine potential benefits of LLTs required that the hardware be synthesized, and implemented with the help of the development tools provided by Xilinx. The implemented hardware was then downloaded onto the ML405 board with 128MB of memory, serial interfacing for connectivity and a Compact Flash card of 512MB for data storage. Furthermore, the LLT is running on a 50Mhz clock as is every peripheral connected to the PowerPC405 through the PLB bus. The experiments were conducted using images extracted from videos and saved as portable pixmaps for processing by the hardware. The images were loaded on the Compact Flash card where they can be accessed by the system for processing. Note that the system could have just as easily communicated, through a memory buffer, with a camera attached to the board. However, the current implementation setup was chosen for flexibility, and control. In evaluating the proposed system, the aim is to showcase the system’s response time to stimuli; in order to do so, this dissertation relies on a Mean-Shift based tracker from Bilkent University [71]. The use of the Bilkent tracker is intended to help us gauge the efficiency of the LLT tracker. The execution time of both trackers over the picture frames were measured by dividing the execution cycles elapsed by 100Mhz, the speed of the PowerPC405. Doing so enables computation of the time in seconds it takes to track an object. The time computed does not take into account the time it takes to load the pictures into the system; but includes printouts and other maintenance code not directly related to tracking.

As per the experiment, that the Bilkent tracker processed frames within 27.2633 seconds. The LLT tracker, on the other hand accomplished the same feat in a fraction of the time notably 0.8791 seconds thereby providing a sizeable improvement in response time to any application that should rely on the LLT. The considerable difference in execution time between the trackers can be attributed to the fact that the LLT is hardware-based. The results support the claim that
providing certain tasks an optimized hardware module can help in improving the reaction time to stimuli of user applications, thusly supporting the notion of bringing sensor network computation to the edge through FPGA-based nodes.

4.2.1.4 Interoperability Interfacing System

The Interoperability Interfacing System (IIS) is the component responsible for managing interaction between nodes in the overlay network and those of the underlying neighboring sub-networks. To conduct its primary function of providing a medium for communication between overlay nodes and sub-networks, IIS must deal with (i) interoperating communication protocols, (ii) data format conversion and optionally, (iii) data aggregation. This is accomplished through coordination among the hardware and software modules incorporating the three aforementioned tasks of the subsystem. IIS is designed with the ability to communicate through various protocols that may be in use by sub-networks in its vicinity. Communication with underlying sub-networks may require IIS to encrypt/decrypt data as per the requirements of such sub-networks. Once contact has been established with surrounding sub-networks, IIS provides user applications with the ability to communicate with such networks, and leverage the available observations and sensing abilities, which are unlikely to be resident in the overlay itself. This ultimately expands the amount of information available to users in accomplishing their tasks. While allowing data exchange between overlay nodes and underlying sub-networks is the primary task of IIS; it also deals with the potential issue that may result from heterogeneous data formats. IIS abstracts the heterogeneity of data formats from users through automatic formatting of incoming and outgoing data streams. In so doing, users can focus on coding the functionalities that they need in their applications instead of dealing with various data formats. Code snippets for format conversions can be eradicated from user applications, potentially reducing their size.
In the prototyped RISN, the IIS performs format conversion between the RGB-24 color format and the YUV color space [72] in the interest of presenting a unified representation of the image data. The conversion between the two formats is performed using two hardware accelerators, with the overlay nodes operating under the RGB-24 format. The hardware accelerators perform conversion of image data between RGB and YUV color formats. The IIS controller manages the accelerators built using the available Intellectual Property Cores from Xilinx to perform the color image format conversion based on the relationship defined in Equation 4-8 [73, 74]. The IIS controller accesses memory by acting as a master on the Processor Local Bus transferring data to and from main memory for processing. The IIS system allows images of size up to 640 X 480 to be converted; thusly requiring at most 2MB of memory to contain the original and converted colors of the images.

\[
Y = R \times 0.299 + G \times 0.587 + B \times 0.114 \\
U = (R - Y) \times 0.877283 \\
V = (B - Y) \times 0.492111
\]

Equation 4-8

The SendData and GetRawData of the appropriate service handler invoke the IIS system from the RISN library in order to maintain format consistency across the hosting RISN node. IIS automatically performs conversion between the two formats depending on the direction of data exchange, with the RISN nodes operating under the RGB format. The format conversion is transparent to users and agent services. For simplicity, the prototype’s IIS communicates solely over Ethernet, although in a real-world scenario, it could be designed to communicate over various communication media and protocols. Within the scope of our prototype, the system is limited to retrieval of image data from sub-networks by the overlay nodes.
Maintaining a uniform data format across overlay nodes is addressed by IIS in providing interoperability. As IIS interacts with various nodes in different sub-networks, it may encounter overlapping or recurrent information from sub-networks. To reduce storage requirements, potentially conserve energy, and avoid processing of repetitive observations, IIS also performs data aggregation, a highly desirable feature even though it is not required. Users have the option of accessing either aggregated or raw data, if the former is available, that the underlying sub-networks have collected. A pictorial representation of IIS and the interaction of its components are presented in Figure 4-4. In short, IIS provides two functions to the remainder of the RISN system: the ability to communicate with underlying sub-networks, and maintaining format consistency.

Figure 4-4: IIS Visualization.
4.2.1.5 Local Sensors

The nodes in the overlay may also sense data from their environments. They are however not required to have any sensing ability and could even exist solely for the purpose of maintaining network connectivity. The format of data sensed by sensors in the overlay must however be in compliance with the data format in use by the corresponding overlay. The prototyped system herein discussed utilizes nodes with no local sensors for simplicity.

4.2.2 Base Station & Underlying Networks

The Base Station performs the same duties as its counterpart in traditional sensor network, in that it essentially manages the network. The Base Station in RISN however, also runs an agent system with a stationary agent: the Base Station Agent (BSA). The BSA serves as an operating interface to users, providing the latter with pertinent information to help locate RISN overlay nodes and their capabilities. Interaction between the BSA and users occurs through the following set of messages:

- LocateNode: Based on the geographic location or service names provided, the base station returns the identity of nodes that could be of interest to the user to allow the latter to accomplish its task
- GetAllServices: Returns a list of all the services available in the overlay nodes. Users can analyze the returned list to determine whether their applications can be supported.
- GetNodeServices: Returns the services associated with a particular overlay node in the overlay network

The set of services used to manage interaction between BSA and users allow the user to locate and identify overlay nodes of interest as well as services available on the overlay node. In
order to accomplish its task, the BSA needs to maintain the list of available services along with location and identity information for the overlay nodes. The Base Station in the prototyped system consists of a workstation with 1GB of RAM and an Intel Xeon processor running at 2.2 GHz running the Aglet server.

4.2.3 User Agents

The users in the system interact with the other entities through the agent interface. In order to accomplish a task, a user contacts the BS and discovers the services available in the network along with the location and identification of potential overlay nodes to which an agent can be deployed. Users can deploy agents to an overlay node in one of two fashions; either directly if the overlay node is addressable from the user’s location, or indirectly by relaying the agent to the Base Station where the agent can then migrate to the overlay node of interest. Note that the Base Station can address every node in the overlay. The deployed agent can then clone itself as necessary to form an ad-hoc agent network, as it carries out its goals. Authenticated user agents must be allowed to traverse the overlay network and the RISN Base Station in search of data of interest for processing. As RISN uses homogenous data format, developers of user agents can focus on specifying the migration pattern of agents along with accessing and processing information from any particular node. While the computing device of the user may be resource-constrained and mobile, the user within the prototyped system resides on the same computer as the Base Station.

The user agent in the prototyped system is concerned with locating a target within the environment monitored by the overlay and underlying sub-networks. To implement the tracker, the user agent migrates to an overlay node where the target of interest is expected to appear initially. The agent would then migrate and clone itself accordingly in the overlay for the purpose
of maintaining and relaying the path taken by the target. Upon arriving to hosting nodes, the user agent obtains the appropriate ServiceHandlers to access services harnessing the capabilities of the underlying components of the system. The ServiceHandlers in turn provide access to the node’s services by interacting with IIS, and LLT through the “RisnNetwork” library. The library provides access to the various drivers, implemented in C, managing the aforementioned components by relying on the Java Native Interface to access customized functionalities provided by the individual nodes in the overlay. Note that in a full implementation of the system, the ServiceHandler would also interact with local sensors through the “RisnNetwork” library.

4.3 System Evaluation

The resources available on any FPGA board are limited. The same holds true for the underlying hardware platform (the Xilinx ML405) herein discussed. The ML405 board contains a Virtex-4 FPGA consisting of 8,544 slices, and one PowerPC405 processor core, used as the GPP. The PowerPC405 processor is set to run at 300MHz with 128MB of RAM. The processor interacts with the peripherals on the system through the Processor Local Bus running at 100MHz. The prototyped RISN node encompasses hardware modules for IIS, LLT, Ethernet, RS-232 serial connection and other system peripherals. The IIS and LLT had to be designed with space limitations in mind. Figure 4-5 showcases the resource utilization that was collected from the Xilinx tools for the LLT subsystem, the IIS converters and the system as a whole. The number of slices available on the board is modeled as “Total Slices Available”. 8215 slices were used to implement the RISN node, of which the LLT system account for 2532 slices and the IIS system accounts for 934 slices.
The proposed RISN system is intended for use in various networking scenarios, however, the prototype specifically targets image-processing applications. The benefits and limitations of RISN are very dependent upon the application and underlying networking infrastructure. A deployment model geared towards determining whether RISN is beneficial towards a particular application and networking system is herein presented (Section 4.3.1). Section 4.3.2 relies on the model to evaluate the RISN prototype.

4.3.1 Strategic Deployment Model

RISN aims at improving data availability to users while increasing responsiveness through hardware accelerators. Determining the trade-offs between implementing a task in hardware or software is application-specific. In general, RISN must be cost-effective to warrant its uses, despite the potential to decrease response time. The system’s cost effectiveness is dependent upon the needs of particular applications and the underlying infrastructure. The use of RISN’s processing model may not always be beneficial to applications, even though increasing
data availability may be desirable in most situations. Through RISN, applications can process
data close to their point of collection; collected data do not have to be relayed to a processing
center. In order to help determine whether RISN’s processing model should be employed for a
particular application, a strategic deployment model is presented, that address the two main issues
of effectiveness: communication load and power. The proposed model analyzes the cost of
utilizing RISN’s processing model versus relaying the data to a processing node.

Cost of communication is a function of the number of bytes, \( B \), that need to be
transferred. Since the data may need to traverse multiple nodes to reach its destination, the
number of hops, \( N \), must also be taken into account. \( C_{Comm} \), as presented in Equation 4-9
represents the cost of sending data from a node to the Base Station, with \( C \) representing the cost
per byte. As per equation 4-9, for any increase in the number of bytes being transferred, the
communication cost increases. At first glance, Equation 4-9 implies that simply reducing the
amount of data that needs to be transferred is sufficient to justify the use of RISN for a particular
application. However, the cost of communication is only one aspect of the system’s cost. By
implementing tasks in hardware, RISN introduces an execution cost in terms of power to the
system. For every LLT implemented, there is an associated cost of dynamic and quiescent power
usage. Dynamic power refers to the energy used by the LLT while it is in use, while quiescent
power refers to the energy used by the hardware module when it is powered, yet inactive. The
energy cost of implementing functionalities in hardware is a function of both dynamic and
quiescent power, but also a function of the number of nodes the application will be using. This

\[
C_{Comm} = NCB
\]

Equation 4-9
relationship is captured in Equation 4-10 where $C_{\text{Exec}}$ is the execution cost, and $M$ represents the load factor or the likelihood that a node’s LLT will be active. $D$ represents dynamic power and $Q$ represents quiescent power.

\[
C_{\text{Exec}} = N[(1 - M)Q + MD]
\]

Equation 4-10

Note that determining whether an application should make use of the RISN’s processing model, is not equivalent to evaluating whether RISN itself should be used in the network. Instead, the deployment model is geared towards helping designers determine whether the data for a particular application should be relayed to a Base Station for processing or should processing occurs in the overlay based on the amount of data to be transferred, and the expected effect of the application on the system’s load factor.

To sum up, the strategic deployment model introduces a mean by which designers can determine whether the improvement in response time presented by RISN can be beneficial to their applications. Further insight on the deployment model can be acquired through the next subsection, which evaluates two tracking applications.

### 4.3.2 Evaluation of the RISN Prototype

Our evaluation of the RISN prototype for target-tracking focuses on the prototype’s communication and power efficiency. Using the strategic deployment model, we evaluate the cost associated with performing target-tracking using the Mean-Shift tracking algorithm based on RISN’s processing model or the traditional approach. The trackers implemented work under the RGB-24 color format and process images of size 640X480. The location of the target is
maintained in both implementations as two 32-bit values relaying the $x$ and $y$ coordinates. For clarity, the two trackers are named RISN_Tracker and Soft_Tracker, with RISN_Tracker being the implementation that uses the LLTs.

By holding the cost associated with sending a byte of data over a network link constant along with the number of nodes, $N$, that the data must traverse, $C_{Comm}$ becomes completely dependent upon the number of bytes that needs to be transferred. Both trackers, as implemented, are solely interested in the location of the target. RISN_Tracker needs to periodically relay eight bytes of data representing the target’s new location. Soft_Tracker, however, must relay the image frames to be processed. As each pixel consists of 3 bytes, Soft_Tracker relays a total of $640 \times 480 \times 3 = 921,600$ bytes, to the Base Station. The number of bytes that Soft_Tracker relays is directly dependent upon the resolution of the cameras in the system.

![Figure 4-6: Communication Cost](image)

Figure 4-6 showcases the theoretical scaled effect of the image size on the communication cost associated with each tracker. The figure shows that RISN_Tracker is
independent of image resolutions, while Soft_Tracker is not. Figure 4-6 demonstrates that considerably less communication cost is incurred by using RISN_Tracker to locate targets of interest.

Evaluation of the system also depends on the cost of execution, which requires knowledge of the system’s load factor, obtained through simulations. The Xilinx tools were relied upon to generate the simulation model of the RISN node, used by ModelSim to generate the Value Change Dump (VCD). The VCD generated is used to estimate the power usage of the system through Xilinx’s XPower estimation tools. The tools estimated the toggle rate at 9.5% for the system as a whole. The toggle rate measures the ratio of time that the system state changes relative to a clock input, and is thus used as the load factor for the system. The dynamic and quiescent power reported for various components of the RISN node (NR stands for Not Reported), are displayed in Table 4-1. Note that Ethernet uses more power than IIS and LLT combined in the experiment conducted.

Table 4-1: Estimated Power Usage

<table>
<thead>
<tr>
<th>Component</th>
<th>Quiescent (Watts)</th>
<th>Dynamic (Watts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>System as a whole</td>
<td>0.36191</td>
<td>1.7188</td>
</tr>
<tr>
<td>IIS</td>
<td>NR</td>
<td>0.00784</td>
</tr>
<tr>
<td>LLT</td>
<td>NR</td>
<td>0.01718</td>
</tr>
<tr>
<td>Ethernet</td>
<td>NR</td>
<td>0.03454</td>
</tr>
</tbody>
</table>

As the RISN nodes are FPGA-based, the included hardware modules, such as LLTs, are subject to quiescent power drainage. The benefits, with respect to execution time, of performing a task in hardware must outweigh the potential drawbacks. In the case of the hardware modules in
RISN, while communication load can be reduced, this must not occur at the expense of increasing the power usage of the system. Using Equation 4-10 and the estimated power usage from Table 4-1, the power consumption of RISN on one node, with hardware accelerators, can be computed as 

\[(1 - 0.095) \times (0.36191) + (0.095 \times 1.7188) = 0.49081 \text{ W}, \]

with 0.095 representing the estimated toggle rate. On the other hand, the cost of execution with no hardware accelerators is 

\[(1 - 0.095) \times (0.36191) + (0.095 \times (1.7188 - 0.01717 - 0.00784)) = 0.48844 \text{ W}, \]

as the application does not use the hardware modules. The amount of energy used for an operation can then be determined, based on the time it takes for the operation to execute. Consequently, three distinct execution times on the prototyped RISN system are measured to conduct an evaluation of the system within the scope of the implemented hardware modules targeting image processing.

\[\text{Figure 4-7: IIS Computational Efficiency}\]

\[PPCS\text{oftTime}\] measures the time it takes a regular application to perform the computation of interest on the prototype. \[Agent\text{SoftTime}\] measures the execution time of a user agent that does not take advantage of the available hardware accelerators. Lastly, \[RISN\_Time\] measures the
The execution time of a user agent harnessing the execution power of the hardware accelerators. The potential agent overhead is highlighted as the difference between \(PPCSoftTime\) and \(AgentSoftTime\). The aforementioned times were measured based on execution of different LLT floating-point operations, such as division, and multiplication, with a varying number of array elements. The execution times of interest for the IIS operation of the prototype were also measured in converting RGB-24 images to the YUV color space.

![Multiplication](image)

**Figure 4-8: LLT Computational Efficiency (Multiplication)**

The results of the experiments are presented in Figures 4-7, 4-8 and 4-9. Figure 4-7 showcases the improved execution time that RISN can provide in maintaining format consistency. \(AgentSoftTime\) converts 180000 pixels in 4.0746 seconds, while \(RISN\_Time\) accomplishes the same feat in 0.5144, an 87.4% reduction in execution time. Figure 4-7 also highlights the negligible agent overhead incurred through the use of RISN’s agent-based computational model.
Figures 4-8 and 4-9 depict the improvement in execution time RISN can provide to applications requiring the specified floating-point operations. For an array of 150000 elements, AgentSoftTime requires 1.576 seconds to compute their square; RISN reduces this execution time by 70.1% by performing the required computations in 0.471079 seconds. The RISN_Time presented in Figure 4-7, 4-8 and 4-9 does not include the time RISN takes to locate the appropriate ServiceHandler in the system, as this overhead, while not a constant, is incurred only once per execution on a node.

By relying on the execution times measured in the experiment, Figure 4-10 presents the energy (in J) used by the system based on the number of array elements being processed. The figure shows that even though RISN has higher power consumption, when the speedup afforded by the hardware accelerators is factored in; RISN actually uses less energy to perform the computation requested.
Lastly, Figure 4-11 shows the theoretical effect of the percentage of dynamic power used by the RISN hardware modules (IIS and LLT). As per the relationship depicted in the figure, the percentage of a node’s power used by the hardware accelerators should be minimized in order to maximize the benefits afforded by RISN, namely, reduction of execution time and energy consumption.

![Figure 4-10: Energy Usage for Multiplication](image1)

![Figure 4-11: Theoretical Effect of Percentage of RISN Hardware Modules’ Dynamic Power](image2)
4.4 Conclusion

This chapter has introduced a novel approach to sensor network that deviates from the traditional view that the network is limited and isolated. Through RISN, existing networks can be made interoperable increase the knowledge set that users can access. The introduction of RISN is geared towards provision of an efficient interoperable and reconfigurable system capable of dynamically adapting to changes in the environment being monitored.

The framework as laid out in this chapter, accomplishes the stated goals through a layered approach in which interoperability is achieved using the IIS systems; the LLTs provide increased computational power and re-configurability; the service-based architecture and the agent system provide dynamic tasking. The improved response time of the system can be crucial to time sensitive operations. The system as a whole, through adaptation of uniform format across RISN nodes, abstracts the heterogeneity of the subnets thereby allowing users the ability to focus on the task at hand while reducing the learning curve associated with the framework. Through the proposed cost model, developers can determine the potential benefits or drawbacks of deploying a RISN-based application. The evaluation discussed showed that the use of LLTs could significantly improve the processing time seen by users, thus allowing for swift reaction to events of interest. However, the evaluation of RISN has thus far focused on performance, the abilities of the framework is showcased in the following chapter through implementation of a distributed tracking application as a proof-of-concept.
Chapter 5

Target Tracking

Target tracking is concerned with maintaining the location of an object, known as a target, as it moves within an environment. There are various methods for determining the current location of the target within an environment. Each method relies on one or more representations of the target in order to locate it within the environment. The representation of the target through its features is intended to allow a tracker to identify the object within the environment. The sensors available to the tracker restrict interaction between the tracker and the environment; as such a camera can support implementations of vision-based trackers, while a microphone could enable acoustic tracking. Furthermore, due to its reliance on sensors, a tracker can only locate a target if the target of interest is within the range of one or more of its associated sensors. Thusly, a tracker may be unable to continuously maintain the location of a target if the target moves out of range of its associated sensors. Improving the amount of non-repetitive sensor data available to any tracker can directly increase the coverage area within which the tracker can locate the target of interest.

RISN has been introduced (see Chapter 4) to allow existing networks to be made interoperable thereby increasing the knowledge set accessible to applications. The system abstracts the heterogeneity of sub-networks, provides an efficient interoperable and reconfigurable system capable of adapting to changes in the environment. Increased data availability, efficient processing and the ability to execute in a potentially heterogeneous environment provided by RISN could greatly benefit applications, such as target-tracking. This chapter introduces the implementation of one such tracker atop RISN as a proof-of-concept showcasing the potential of the RISN framework to support applications in achieving their goals.
despite potential heterogeneity of the network. The background related to our work has been mainly discussed in section 2.3. Section 5.1 will further dive into the topic, focusing on vision-based tracking. The architecture of the agent-based tracker is introduced in section 5.2; the implementation of the application is discussed in section 5.3, and conclusions are drawn in section 5.4, highlighting the contributions of the work.

5.1 Background

Maintaining the location of an object as it moves within an environment of interest constitutes the primary task of trackers. Section 2.3 discussed the state of the art proposals for tracking within the sensor network environment. Such proposals have assumed the existence of binary trackers or do not restrict the system to any particular sensory input. While the herein presented work is applicable to environments with a vast array of sensory input, the proof-of-concept scenario specifically deals with vision-based tracking. Vision-based tracking aims at generating an object’s trajectory through analysis of the objects location in successive frames of a video. [66] categorized the methods used to track objects as Point, Silhouette, or Kernel based [66] depending on the representation of the object by the tracker. One such method for locating an object is a non-parametric mode-seeking algorithm known as mean-shift. Mean-Shift performs hill-climbing to locate the nearest local mode of a kernel density estimate that can be used to represent a target. Regardless of how the tracked object is represented, occlusion, referred to as the obstruction of the tracked object by other objects in the environment [66], can occur. Common approaches to resolving occlusion includes location prediction, silhouette projection and even implicitly through generalization of object tracks [66, 75, 76]. Furthermore, multi-view based tracking have also been proposed in the literature [77-79], and is defined as a subset of kernel-based tracking [66]. Multi-camera based tracking refers to the use of multiple cameras in
determining the location of the tracked object. While essentially relying on the same principle as multi-camera tracking, the proof-of-concept is however an autonomous agent-based distributed tracker capable of subsisting in a potentially heterogeneous sensing environment while leveraging the limited coverage of any one sub-network.

5.2 Autonomous And Distributed Target-Tracking Architecture

As a proof-of-concept of RISN’s ability to ease development of distributed applications capable of executing in heterogeneous environments, a distributed target-tracking application will be introduced, independent of the RISN_Tracker discussed in chapter 4. The application continuously maintains the object’s location despite the potential heterogeneity of the network and limited coverage of any one sensor or sub-network.

5.2.1 Tracking System Architecture

With the assumption that the object of interest is the only mobile physical entity in the environment, target tracking must inevitably deal with the issue of the object moving out-of-range of one or more sensors or an entire isolated network. The potential heterogeneity of the sensors and their spatial deployment, their limited field of view, and the potential for unexpected occlusions of the object of interest greatly complicates the task of determining the target’s current location. However, by restricting the primary function of a tracker to that of returning the location of an object in space, a distributed tracker can be independent of any one tracking algorithm implemented on a contributing sensor node. The distributed tracker can instead simply rely on the perceived location relayed by each sensor. RISN aims at leveraging data available from sub-networks, abstracting the heterogeneity of such networks while providing support for dynamic
tasking and efficient processing (see Chapter 4). RISN is thus an attractive platform to develop a distributed tracking application capable of accomplishing its task despite the possibility of the object moving out-of-range of any one sensor or network as well as the heterogeneity of nodes’ sensing abilities and required feature set used in tracking. Thusly, implementing one such tracker can serve as a proof-of-concept of RISN aforementioned abilities. In so doing, the proof-of-concept tracker relies on two main components, namely an Ad-Hoc Agent Network (AHAN), and a Tracking Service Handler (TSH) for each possible contributing RISN overlay node.

The two components (AHAN and TSH) interact to allow the object of interest to be located as it travels through the network. The system works by deploying an agent to the overlay node, with coverage of the location where the object of interest is expected to appear initially. The agent contains a Target field that represents the object being tracked. The deployed agent uses the appropriate representation of the object as specified in the Target field of the agent, to determine the current location of the object through communication with the TSH of the overlay node. Using the returned location, the agent clones itself and dispatches the clones to overlay nodes with sensing coverage of the target’s perceived path as determined by TSH. The initial agent and its clones form the AHAN, which is described in the next section.

5.2.1.1 Ad-Hoc Agent Network

The Ad-Hoc Agent Network (AHAN) is made up of coordinating agents dispatched to locations of interest in the RISN Overlay. The network determines the current location of the target based on the individual tracking results received. Target maintains past locations of the object and the possibly diverse set of features that may be used to locate the object. To illustrate the latter point, consider that an object, within the scope of computer vision alone, can be represented as parametric and non-parametric probability densities, as well as templates formed
through the use of geometric shapes to cite a few [66]. Furthermore, various other features such as heat and sound signatures can also be used to represent the object. To deal with such a vast set of possible feature representations of an object, Target is represented as a class capable of maintaining various representations of the object of interest, each of which is accessible through their statically pre-defined names. The appropriate representation of the object can be retrieved in order to determine the object’s current location on any particular overlay node by using the `retrieveObjectModel(String modelName)` method of Target. Note that there is an underlying assumption that Target is initialized with every possible representation of the object that might be required by TSHs available on RISN nodes. By relying on the perceived path of the object being tracked, clones of the initial agent are dispatched to appropriate neighboring overlay nodes forming the AHAN. The clones and the initial agent communicate through the following messages:

- `cmdGetTargetLocation`: instructs clones to determine the current location of the target of interest using the sensor data streams managed by the clone.
- `DestinationAddress`: specifies the address to which the clone should migrate to and which sensor data stream it will manage at the destination.
- `cmdTerminateClone`: terminates execution of the receiving clone and frees up used resources.

The initial agent migrates through the network as the object moves; clones are terminated when they are no longer capable of helping in determining the current location of the target. The initial agent aggregates the retrieved locations of the object for the current iteration, and uses the aggregated location as the current location of the object observed by the system. As mentioned earlier, the agent network is built starting with an initial agent, executing on a node from which the target is assumed to be locatable. The initial agent interacts with the TSH in order to maintain the location of the target. Details regarding the interaction between the agent network and the TSH are provided in the following section.
5.2.1.2 Tracking Service Handler

The Tracking Service Handler (TSH) abstracts the potential heterogeneity of tracker implementations or associated feature sets, thereby allowing for the object to be located using the appropriate tracker or suitable features for the current overlay node. TSH is a RISN service dedicated to tracking that is available through a handler. In this implementation, TSH is initialized by specifying the size of the 2-D virtual space representing world coordinates, being monitored as well as specification of the sensors available on the overlay node along with the address of neighboring overlay nodes, with which the TSH can communicate. For each sensor on the local overlay node, the handler is assumed to maintain the appropriate, and possibly optimal tracking algorithm for the sensor. The handler also determines and maintains the Field Of View (FOV) of each local sensor. Details regarding the implementation of a sensor’s FOV are provided in section 5.3. The FOV covered by sensors of neighboring overlay nodes is retrieved through communication with remote SSAs. In essence the handler allows execution of the following commands:

- **GetNodeFOV**: retrieves the FOV of a particular sensor stream available on an overlay node.
- **cmdFutureAddressesOfPoint**: returns the address and identifiers of known sensor streams whose FOVs intersect with the specified point.
- **cmdFutureAddressesOfSegment**: returns the address and identifiers of known sensor streams whose FOVs intersect with the specified segment.
- **cmdFutureAddressesOfLine**: returns the address and identifiers of known sensor streams whose FOVs intersect with the specified line.
- **cmdFOVIntersectPoint**: returns true if the FOV of the specified sensor intersects with the specified point.
• *cmdFOVIntersectSegment*: returns true if the FOV of the specified sensor intersects with the specified segment.

• *cmdFOVIntersectLine*: returns true if the FOV of the specified sensor intersects with the specified line.

• *cmdTrack*: retrieves the required feature representation of a target and attempts to locate the current location of the object in the neighborhood of the target’s last known location. The new location of the object is returned, without any mapping to the virtual space. *cmdTrack* further requires the specification of which sensor to use in tracking the object. Specification of the sensor allows TSH to determine which feature representation to extract from the *Target* field.

• *cmdMapFromVirtual*: maps the specified location in world coordinates to the sensor’s local coordinate indicated on the overlay node.

• *cmdMapToVirtual*: maps the specified sensor’s local coordinate to the system’s world coordinates.

Details of the interaction between the components of the distributed tracker are provided in the next section as an implementation of the tracker is presented.

### 5.3 Tracking System Implementation

The tracking system herein implemented relies on the RISN framework introduced earlier. As such, introduction of the tracking system implementation is meant to highlight RISN’s ability to leverage data from isolated networks, intelligently process sensor observations and abstract the potential heterogeneity of underlying networks.
Figure 5-1 presents the layout of the objects in our experiment; the system consists of 2 RISN Nodes and 8 cameras positioned across a room. The object to be tracked in the room is a red ball. The choice of a ball is motivated by the fact that its representation is fairly independent of the viewing angle. The proposed distributed tracker is implemented through the use of the 2 RISN nodes (Node-1 and Node-2) attached to the base station from our earlier experiments. The nodes implement the Tracking Service Handler (TSH) discussed earlier, and an agent system to support migration of user agents, herein referred to as application agents, tasked with locating targets of interest through interaction with the TSH. Communication between the nodes and the base station occurs over Ethernet. Each RISN node manages 4 of the 8 cameras in our experimental setup. Node-1 manages Cam1, Cam2, Cam6 and Cam4; while Node-2 manages the remaining cameras. The 2 nodes embody nodes in the RISN overlay that the distributed tracker depends on in accomplishing its task. The reason behind this setup is to emulate an overlay
network consisting of the RISN nodes managing two potentially isolated and heterogeneous subnetworks with overlapping coverage areas

\[
M = \frac{W(1:N)}{W(N+1)} \quad \text{where} \quad W = [U \quad 1]^* H^T
\]

Consequently,

\[
U = \frac{W(1:N)}{W(N+1)} \quad \text{where} \quad W = [M \quad 1]^* H^{-T}
\]

Equation 5-1

The cameras in the system do not share the same resolution, and act as sensors of the subnetworks controlled by nodes in the RISN overlay. Mapping of individual camera coordinates is performed using the homographic matrix of the cameras. The mapping to and from world coordinates is performed using (Eq. 5-1) [80, 81]; where H represents the homographic matrix of the camera, M is a 1-by-N matrix representing the world coordinates and U is a 1-by-N matrix representing the local coordinates of the camera. The homographic matrix of Cam2 is displayed in (Eq. 5-3). The cameras were calibrated offline and their homographic matrix, mapping individual camera coordinates to world coordinates, were computed and loaded onto the appropriate TSH.
The TSH of each RISN node relies on the homographic matrices to perform bi-directional mapping from the local coordinate of a camera onto the system’s world coordinates. The TSH maps the pixel coordinates of any managed cameras. In so doing, application agents only need to deal with the system’s world coordinate, as the TSH abstracts the heterogeneous coordinate systems of each camera. The world coordinates of the pixels of Cam2, $M_{Cam2}$, are computed through the use of (Eq. 5-1) and $H_{Cam2}$ as shown in (Eq. 5-2). The TSH also uses the homographic matrix to determine the FOV of each camera that it manages. The FOV is computed as a mapping of all the pixel coordinates of any particular camera onto the world coordinates that the system monitors. The FOV is thus computed by mapping the image coordinates, $U$, of each pixel of a camera onto its corresponding world coordinate, $M$ using (Eq. 5-1) and the homographic matrix of the camera being processed. The FOV of the cameras in the system, are mapped onto a two dimensional “virtual” space representing the $X$ and $Y$ world coordinates being monitored. TSHs maintain the FOVs of managed cameras as a Java BitSet object from the Java Library, to determine whether a vector, a line or a point intersects with the FOV of any particular camera.

Once the handler is setup, having acquired the FOV of neighboring sensors, the application agent is deployed on Node-1, which maintains Cam2. The application agent, upon
arriving at Node-1, initializes the target based on the specified location of the target in the initial frame. The application agent at the initial node relies upon the algorithm presented Figure 5-2 in order to track the target’s location over the successive video frames. The target in question is a red ball represented by its color histogram made up of 16 bins per color channel in the RGB color space. For the purpose of experimentation, we work with a target whose representation is independent of the viewing angle of any one camera, thereby abstracting issues that may arise due to the fact that the histogram representing an object can be very different depending on the viewing angle used to compute the histogram.

```
distributedTracker ( WLoc, Target )

    WLoc = last known location of the object in world coordinates
    CLocation = location of the object in image coordinates
    Target = Object representing target to be tracked
    cloneLocations = set of Locations of the object as determined by neighboring nodes
    neighbors = neighboring RISN nodes, and associated sensor streams

    {
        FOR EACH Frame to be processed
        {
            CLocation ← mapToImageCoord(WLoc)
            IF ((CLocation = trackTarget(CLocation, Target)) THEN
            {
                WLoc ← mapToWorld(CLocation)
                Target.updateMotionVector(WLoc)
            }
            ELSE
            {
                neighbors ← findNeighbors(Target.getMotionVector())
                cloneLocations ← ActivateClones(neighbors)
                WLoc ← Aggregate(cloneLocations)
                Target.updateMotionVector(WLoc)
            }
        }
    }
```

Figure 5-2: Distributed Tracking Algorithm

The distributed tracking algorithm works by iterating over the different numbered image frames. For each frame and the last known location of the object, the application agent attempts to
locate the target locally. In the event the object is not visible locally, the agent determines the sensors and the managing overlay nodes that can be used to track the target. If there is a clone managing a suitable sensor on a neighboring node, a message is relayed to the clone asking for the updated location of the object. A new clone is dispatched, in the event one does not already reside on the nodes of interest.

Figure 5-3 presents a pictorial representation of the FOV of all eight cameras available in the system, with that of Cam2 highlighted in green. The figure displays the locations of the object as tracked by the dispatched application agent based on the location of the object in the initial frame, represented as F1 in the figure. From the first frame, F1, to the 27th frame (F27), the object is visible in the FOV of Cam2. On the 28th frame however, the application agent must rely on the other cameras in the sub-network of Node-1 to maintain the location of the object. It is worth noting that at frame 43 (F43), the target leaves the FOV of all cameras under the control of

![Figure 5-3. Overview of Tracking System](image)
Node-1. In order to maintain the location of the object, as per the experimental setup, the application agent relies on data from neighboring networks, by dispatching an agent to Node-2 based on the expected path of the object and its intersection with the coverage area of the sensors managed by Node-2. In the experiment, this resulted in the system tracking the object using Cam3, Cam5 and Cam8 from Node-2 by utilizing, and processing data from neighboring networks to accomplish a common goal.

The initial location, (F1), of the object being tracked as seen by Cam2 is displayed in figure 5-4-a; while figure 5-4-b shows the location of the object seen by Cam2 after 27 frames (F27) have been processed. On the 28th frame (F28), the object is no longer visible by Cam2; however, it is still visible by Cam1 in the same sub-network. In the 43rd frame (F43), the object
also leaves the FOV of Cam1, thereby becoming invisible to Node-1. The tracker is able to maintain the location of the object in world coordinates, despite the fact that it is no longer visible from the initial camera or sub-network. When the object reappears in frame 62, (F62), it is accurately located by the application agent using Cam2 as shown in figure 5-4-c, by relying on the information harnessed from the other sensors in the system. In frame 68 (F68), the object also becomes visible to Cam1.

As the handlers for each camera is responsible for implementing the suitable target-tracking algorithm, the system makes no assumption of the homogeneity of each tracker, as a result, the trackers could be completely heterogeneous. For simplicity however, the herein discussed implementations rely on the Mean-Shift algorithm. The only requirement is that the

Figure 5-5. Bandwidth adjustments
Target is able to supply to the handler the necessary parameters on which to operate by using the
`retrieveObjectModel` method. Although the representation of the object is assumed to be independent from the sensors’ point of views and that the sensing devices in the system consist solely of cameras; the size of the object however varies depending on its distance from any particular camera. Thusly, it is important that the TSH of each node is able to maintain and adjust the size of the object from each camera independently. This is accomplished by adjusting the bandwidth parameter of each Mean-Shift tracker using the method proposed in [67, 68]. Figures 5-5-a, 5-5-b, and 5-5-c showcase how the TSH adjusts to the changing size of the object in successive frames from Cam7.

### 5.4 Conclusion

Chapter 5 has discussed an autonomous distributed target tracking application that is independent of the sensing abilities of any one node that can be implemented on the RISN system (see chapter 4). The tracking application discussed simply requires the presence of a Tracking Service Handler, which can abstract the sensing heterogeneity of nodes, along with the presence of all possible feature set representation of the object of interest, in order to maintain the continuous locations traversed by the object as it moves in and out of the view of any nodes or isolated network. The TSH maps the local coordinates of cameras used in the experiments to world coordinates by relying on the homographic matrix of the cameras that has been computed offline.

The tracking application discussed has been implemented as proof-of-concept of RISN ability to support distributed applications capable of subsisting in an heterogeneous environment. By relying on agent technology, the tracking application possesses the ability to intelligently
process data from surrounding sub-networks made available through the RISN framework. The work, as discussed, relies solely on vision-based tracking, however, through the TSHs, it is conceivable that other sensory inputs, such as acoustic, could be used to track objects of interest using a more diverse set of features.
Chapter 6

Conclusion & Future Directions

This dissertation aimed at tackling some of the issues stemming from the prevalent view of sensor networks as a system consisting of static nodes forming an isolated network. Such issues include the use of a static system subsisting in a dynamic environment, isolated networks with potentially overlapping coverage area thereby suppressing the ability of applications to obtain a more comprehensive view of the environment. To address the aforementioned issue, this dissertation proposed the use of RISN, an approach to sensor networks that aims at allowing existing sub-networks to interoperate, while granting applications the ability to efficiently harness and process data. RISN relies on mobile agent technology to foster an interoperable view of sensor network with reprogrammable nodes. RISN supports agent-based user applications that migrate through the overlay and take advantage of data leveraged from sub-networks. Furthermore, RISN takes advantage of FPGA devices to increase the processing power of nodes in support of computationally expensive pre-defined tasks. To support efficient processing of collected data, RISN allows user applications to take advantage of efficient implementations of tasks with which the user is expected to deal with through the use of services. The platform abstracts the heterogeneity of sub-networks by maintaining a consistent format in the overlay network. One stated goal of sensor networks is to occupy an area with minimal disturbances to the environment and its occupants. With the deployment of a new network for every new task, this goal is fated to be breached by leading to proliferation of nodes. With RISN’s ability to interoperate with other networks, the number of networks with similar sensing abilities that need to be deployed in an area can be greatly reduced, as the system facilitates interoperation and leverages available resource for processing by application agents.
As RISN supports agent-based user applications, it is imperative that the system deals with the potential security issues inherent in the agent paradigm. To that end, this dissertation has also tackled agent security through the introduction of a distributed and adaptive security-monitoring framework achieved through agent collaboration across multiple hosts. The proposed framework has been implemented for the Aglet agent platform, but can however be applied to any agent platform. The monitoring framework relies on the idea of boosting, and attempts to protect hosts by utilizing the reputation of agents obtained from coordinating hosts based on the agents’ execution patterns. The system is capable of supporting the use of various independently trained classifiers. Implementation of the monitoring framework in Aglet further required that we improved the security of the Aglet server through provision of secured communication, ability for agents to detect tampering of their data, and allowing hosts to restrict the actions of malicious agents that may lead to denial of service attacks.

Agent-based applications enable moving the computation close to the point of collection of data in the RISN network. RISN further provides applications with hardware accelerators to support commonly used tasks by implementing the RISN nodes on FPGA. This dissertation has shown that the use of accelerators can increase the execution speed of applications and could also reduce the energy requirements of such applications. The use of FPGAs also leaves open the possibility of reprogramming one or all the nodes in the network based on changing requirements, as may be the case when a new sub-network is within range. Interaction with sub-networks is handled by the IIS sub-component of the RISN framework, allowing for interoperability while leveraging the data collected from an area for efficient processing by agents executing on the node.

This dissertation has also discussed the implementation of a distributed tracking application designed as a proof-of-concept of RISN’s ability. The tracking application is restricted to vision-based trackers and maintained the location of an object as it traversed a set of
isolated networks whose data has been leveraged through RISN. The application introduced is an autonomous distributed target tracking application that is independent of the sensing abilities of any one node in the system. The tracking application discussed simply requires the presence of a Tracking Service Handler, which abstracts the sensing heterogeneity of nodes, along with the presence of all possible feature set representation of the object of interest, in order to maintain the continuous locations traversed by the object as it moves in and out of the view of any node or sub-network.

While a RISN prototype has been implemented and discussed throughout this dissertation, further work needs to be conducted to improve the system. The following represent a succinct list of research directions that can be pursued in the near future:

- Investigating whether RISN can increase the lifetime of one or more power-starved networks can be valuable in fostering adoption of the system.
- This dissertation has assumed that the communication protocols and data formats of underlying sub-networks are known in advance. Locating existing sub-networks in the surroundings of the RISN overlay, as well as reconciling heterogeneity of protocols and data without a priori knowledge of their exact nature need to be automated to closely mirror real-world scenarios.
- Methods for communicating encrypted data and preventing compromise of the encryption keys in the course of “discovering” new sub-networks should also be investigated to further strengthen the security mechanisms of RISN-based systems.
- This dissertation has only considered wired Ethernet in its evaluation of the RISN prototype. Evaluation of the system under the constraints of wireless communication, which is expected to complicate the challenges of deploying RISN-based systems, needs to be undertaken.

Ultimately, RISN is expected to grow into a robust, flexible, and efficient system capable of adapting to the dynamic needs of applications. RISN can be crucial in supporting ad-hoc
applications possibly subsisting in a time-shared environment that provides the necessary sensing data in an area of interest for efficient processing thereby reducing the potential for sensor proliferation. RISN could be envisioned supporting an ad-hoc search and track application from law enforcement authorities to locate dangerous criminals. Such an application could rely on RISN to dynamically communicate with underlying networks, and leverage the data from available sensors deployed in the area of interest. Through reliance on RISN, the application could locate and direct law enforcement officials towards the safe apprehension of the suspected criminal. With the ubiquity of cellular phones equipped with cameras and other sensors, there is a high potential for vast amounts of data to be collected and processed. Pervasive systems can greatly benefit from access to such data sets, and RISN is a step towards making this vision a reality to enable autonomous applications that are impervious to the underlying heterogeneity of the system.
Appendix

Agent Security Addendum

1 Communication Vulnerability Analysis

Of primordial importance to any agent systems is the ability to allow agents to communicate and roam the network. Such communication and mobility must be performed in a secure manner. As mobile agents, Aglets suffer from the same security vulnerabilities that plague the programming paradigm. The interception of Aglets as they move from one host to another is a serious threat encompassed by the need to protect agents from malicious hosts. Protecting Aglets against malicious hosts entails ensuring that the agents and their messages are sent exclusively to the intended entities (hosts or agents).

Presently, the Tahiti server supports authentication of the entities that wish to establish communication, typically the authentication of servers. The communicating parties in Tahiti are authenticated using a Challenge-Response scheme based on the Diffie-Hellman algorithm, a cryptographic protocol, which allows two parties to exchange a secret key over an insecure medium without any prior secrets. The authentication of the entities is done in phases once one party has initiated the cycle, and is based on security domains defined as a region with homogeneous level of security.

In our attempt to determine the efficiency of Tahiti in protecting the communication channels used by Aglets, we established a link between two instances of the server running on different machines. Our goal is to try and intercept the packets being exchanged between the two servers, as the interception of such packets can easily lead to the reconstruction of the Aglet or message being transmitted. We used a third machine and the readily available Dsniff software to
try and intercept the packets between the two servers. As a result of our experiment, we were able to capture the packets containing the Aglet in its serialized state as it is being transmitted over the unsecured communication channel. The ease of conducting such an attack is not surprising, as Tahiti does not use any encryption during communication. The authentication scheme, in place in Tahiti, is used only to verify the identity of the servers but not as a prelude to a key exchange that would be used to encrypt future communication. The authentication scheme was designed to prevent security attacks, such as the reflection attack, against the communication layer of Tahiti.

Our experiment shows that the authentication scheme is still susceptible to more complex attacks such as man-in-the-middle (MITM) attacks carried out through eavesdropping on a communication channel with the ability to modify or insert data into the channel. It also exposes a weakness in Tahiti in that Aglets can be intercepted while traveling on the insecure communication channels and be made to execute on a malicious host. Our first experiment has thus led us to conclude that the Aglet framework cannot currently satisfy the requirement of agents to migrate exclusively to intended hosts.

2 Data Vulnerability Analysis

Protecting mobile agents against malicious hosts must occur on two different levels. Agents must first be able to freely roam the network and travel only to intended hosts; moreover, agents must be able to freely execute on the hosts onto which they have migrated. To this day, there exists no highly acclaimed solution to the problem of protecting an agent from a malicious host, although numerous proposals have been submitted to address the issue of identifying malicious hosts (see section 2.1.1). When an Aglet arrives at a host to perform a computation, it is at the complete mercy of the host in question. Hosts are capable of manipulating the Aglet’s data collected from previous hosts. Tahiti presently has no mechanisms in place to help detect
tampering of an Aglet’s data from a malicious host nor does it provide any mechanism to help identify the existence of such hosts. It is thus imperative that such issue be addressed to foster the use of agent in privacy-aware applications.

3 Resource Vulnerability Analysis

The ability for hosts to support the execution of potentially malicious agents has been well discussed in the literature within the scope of mobile agents in general. Aglets suffer from the same limitations in that malicious Aglets could be detrimental to the proper functioning of a host. Being that Aglets are built on top of Java technology, the Tahiti server makes adequate use of the Java sandboxing techniques to protect hosts. Aglets execute within a runtime environment, which controls their access to system resources such as files and devices. Aglets are allowed to execute a set of actions on themselves that allows them to move from one execution state to another during the Aglets lifecycle.

During the course of our investigation, we have identified an inherent vulnerability of host resources stemming from the normal lifecycle of Aglets. To illustrate our observation, we implemented an Aglet application whose purpose is to attempt to consume as much resources as possible on a host through traversal of the Aglet’s normal lifecycle stages thereby initiating a Denial of Service attack. To ascertain the effect of such a behavior from an Aglet, we considered the best-case scenario in which the clones of the Aglet do not carry out any computation, nor does the Aglet itself carry any computationally expensive operations. The developed Aglet application was instantiated from a Tahiti server running on a 1.0Ghz G4 processor with 1.0GB of memory. The application that we developed carried out the attacks on the host through repeated cloning, as well as by creating and dispatching new Aglets to a target host. The attack was successfully carried as well through the activation or retraction of Aglets. The running instances of the Aglets
had a dramatic effect on the amount of memory being used by the Tahiti server. On average, with approximately 7000 instances of the Aglet application, the server reports a java.lang.OutOfMemoryError and is then unable to process any more requests including the creation of new Aglets.

This has led us to conclude that Tahiti can be easily exploited by a malicious Aglet or even by an Aglet that has been improperly designed. The erroneous introduction of an infinite loop (or even a loop with a large number of repetitions) around a statement that would cause the Aglet to clone itself, or create an instance of another aglet could lead to disastrous effects on the resource utilization of a host.

4 Privacy-preserving Information Retriever

We have prototyped a Privacy-preserving Information Retriever (PIR) to highlight our contributions in securing the Aglet execution environment. The PIR prototype is based on MAMDAS, a multidatabase access system tailored to support user mobility. PIR has been prototyped to model the scenario in which companies need to collect information and make a hiring decision about an individual. Government agencies along with private corporations maintain pertinent data that are, for the most part, readily available to the public. Arrest records, credit report as well as past salaries represent some of the information that may come into play in hiring a potential employee. The PIR has its own set of security requirements. As we explore such requirements, we will demonstrate the mechanisms available in SAS to provide the required level of security.

Being built on top of MAMDAS, PIR makes heavy use of agents to collect the information of interest. Notably, PIR uses an Aglet, DataSearchWorker (DSW), to roam a hierarchical network structure that reflects the semantic relationship of the datasets. Due to the
potential sensitivity of the data collected by the DSW, the Aglet need to only travels to hosts of interest to the task at hand to ensure reliability of the data. As we have shown earlier, third parties can intercept agents in Tahiti, thereby compromising the security of PIR. On the other hand, through the use of SSL, SAS provides applications with the necessary mechanisms to ensure that delegated agents only travel to intended hosts. The prototyped PIR makes use of the available mechanism to attain the communication security needed to prevent access to pertinent data through authentication of the entities in the network.

To collect the required data, the DSW will visit various hosts. The hosts visited may be past employers of the hiring candidate, and thus may benefit from sabotaging the candidate. If the candidate’s records are altered, such as through insertion of arrests records or increase of past salaries, the collected data cannot be used to make a reliable decision. PIR requires that the integrity of the collected data be maintained as it will be used in a manner that will affect the future of the company. While Tahiti does not provide any support to applications with similar requirements, SAS provides the basic functionalities to allow detection of data tampering. The PrivateData library allows applications to profit from such functionalities. Through the use of the library, PIR allows users to verify the integrity of the result, as the DSW requires hosts to sign any collected data. If any of the collected data has been corrupted, PIR can notify the user, should the latter decide to go through with the verification step.

Aside from communication and data security, equally important to PIR is the issue of ensuring that the resources of hosts are not consumed in vain. Malicious agents, through lifecycle operations, may subject hosts to DoS attacks; such occurrence would prevent PIR agents from accomplishing their goals and collect the information of interest. Within PIR, multiple agents may need access to the same host, as some personal information, such as credit reports or criminal records are on a limited set of hosts. Similar to any application, data availability is of primordial importance in our prototyped application. The existence of the system would be futile
if it could not collect the criminal records of a potential employee. Tahiti, as we have discussed, can be subjected to such DoS attacks as it strays away from the micro-management of Aglets. SAS addresses the issue through the introduction of a MonitorAglet to track the actions of agents in the system. The MonitorAglet ensures the availability of DSW’s hosts of interests by limiting the number of instances of agents.

The PIR prototype has been achieved in SAS due to the fact that SAS provides the basic functionalities required to sustain a secure system. PIR allows users to specify the global term to use in the search along with the semantic specifications. Once the results are collected, the user can choose to verify their integrity. SAS allowed the agents in the prototyped application to travel only to intended hosts through authentication; and data availability is provided through the MonitorAglet.
Bibliography


[73] Anonymous "YCrCb to RGB Color-Space Converter v2.0,"

[74] Anonymous "RGB to YCrCb Color-Space Converter v2.0,"


VITA

Evens Jean
470 Waupelani Drive # 207
State College, PA 16801
347-886-6558
jean@cse.psu.edu

Education

Ph.D., Computer Science & Engineering
The Pennsylvania State University, Degree Expected Spring 2011

Bachelor of Science, Computer Science
City College of The City University of New York, New York, February 2003

Publication and Presentations


